Financial Surveillance Using Big Data

Project CS 2017

Uppsala University

Fatimah Ilona Asa Sabsono
Daniel Edin
Filippos Petros Lanaras
Emanuel Lind
Martin Matus Saavedra
Michael Wijaya Saputra
Rahul Sridhar Setty
Satya Vrat Shukla
Ludvig Strömberg

January 12, 2018

Abstract

Modern stock market trading is now a number of automation techniques and generates extremely large amounts of data that need to be processed and analyzed concurrently. There is an urgent need in providing market surveillance which can handle this data in a systematic and timely manner without incurring heavy cost. A cloud based approach to read and parse the data is the next step in development. In this project, we have utilized Apache Spark to create an application that processes the data and have provided the tools required for running pattern detection so as to find out suspected cases of fraud. Also, point out anomalies in the data generated by Scila with the help of machine learning techniques.

Acknowledgements

All of us collectively as a team would first of all like to thank Uppsala University and Scila AB for providing us with a unique opportunity to work on a real, industrial project. In a fast developing field like financial technology(fintech), being able to on real, legitimate problems and figure out solutions for them, was a learning period that we greatly benefited from. Financial technology is an important field that continues to grow at an exponential rate and so does its impact on the wider world. Hence the need for carrying out market surveillance in this interconnected world only grows. To have been able to look at the difficulties that arise in doing so in a scaled up manner and exploring new technologies and a more cloud based approach for the same, was an invaluable chance for all of us.

One of the most fulfilling part of doing this project was being able to work with all the various members of our team. We come from a wide range of backgrounds, but right from the beginning we adopted an approach of open and clear communication that allowed us to approach our task in a systematic manner. We utilized the methodology of scrum to divide the project in simple, yet effective chunks which we then managed to complete in a timely manner.

We would be remiss to not mention the support provided by our project course coordinator, Edith Ngai, and the course assistants, Amendra Shrestha and Xiuming Liu, for their invaluable efforts in securing for us the support we needed from the University in providing the workstations and the rooms which worked as our office. Having a proper work environment, and the break room, allowed us to approach our work in a professional and timely manner.

Fredrik Lyden, Gustav Tano, and everyone at our industrial partner, Scila AB, were always ready to take time out of their busy workdays to clear our doubts, whether by coming down to Uppsala for demos, or by interfacing over emails and skype. Their almost ready availability allowed us to proceed with the tasks per our plan and for that, we would like to thank them.

We would also like to thank Mikael Lundgren for coming at the beginning of our project and giving a lecture on project management, the importance of agile working methodology, and maintaining a daily scrum schedule.

Contents

List of Figures

1 Introduction

1.1 Finance Surveillance

The ever increasing interconnected nature of global stock markets has not just contributed to an unprecedented growth of capital, it has also led to a faster flow of market participants from one market to another, as per changing market forces. [46] Worldwide, many billions of transactions take place on a daily basis and there is an ever-increasing need to track and identify rogue actors who connive to carry out fraudulent activities. There are more participants present in the markets than any time earlier, and consequently, larger the chances for fraud to exist. The need for proper surveillance has therefore never been greater. [51]

Along the way, trading in modern stock markets has now grown to incorporate a large amount of automation, which must lead to an explosion in the number of datasets being generated. These datasets require concurrent processing and analysis in a systematic manner so as to ensure all the various regulations across different markets are followed while ensuring that not too heavy cost is incurred.

1.2 Big Data

A cloud-based approach to reading and parsing the data is the next step in development since the amount of data being generated is so immense, it is referred to as Big Data, a colloquial term that has now transitioned into being a formal one. On a daily basis, there is an almost inconceivable amount of data is generated. It is not just the amount, it is the speed at which it is being generated and the varying number of types of data; formatted, unformatted, structured, unstructured.

As markets generate more and more complex data, the importance of storing and analyzing the complex data sets, only increases. Hence doing so by utilizing the tools and techniques used for Big Data is the next logical approach. For implementing market surveillance functionality, the traditional approach of utilizing large servers does not compare favorably with big data methodology that utilizes clustering of simple hardware that provides for easier scaling options and parallelizes the data over the clusters.

1.3 Project CS and Scila AB

This report provides the documentation for a prototype application developed by students who undertook the project computer science course at Uppsala University. The course provides the students with an opportunity to gain experience in running a big project, right from the planning to the completion stage, how to go about constructing a complex distributed system and to give hands-on experience on modern construction principles and programming methods

This year, the project was set up in coordination with Scila AB, a Stockholm based financial technology company, and Uppsala University. Scila provides trading surveillance products built on many years of experience from both market surveillance and systems design. Scila Surveillance uses modern technology to give the customer a seamless route from detection of market abuse to presentable evidence. Scila delivers the future of modern market surveillance technology by offering trading venues, regulators and market participants the most competitive solution available.

1.4 Project Goals

The main aim [52] of this project was to create a prototype application that read a large amount of financial data that was produced by the Scila system and provide cloud-based tools to that would:

- Process the data in a cloud environment
- Batch-oriented market abuse pattern detection
- Anomaly detection using Machine Learning
- Batch/ad-hoc visualizations/reports

2 Background

2.1 Financial Market Surveillance

2.1.1 Why is it necessary

As financial markets shift even more towards automated trading and involve such techniques as high-frequency trading, where markets get millions of orders in minutes, the need for surveillance in financial markets has only grown. Moreover, markets worldwide are heavily interlinked and there is a constant flow of capital through the various markets which allows for trading to occur every hour of every day. To account for the rapid speeds with which transactions are placed, canceled, and updated, and ensure that they are in line with the various regulations and rules in all the markets, strict financial surveillance is extremely important.

2.1.2 How it Happens

There are various methods of market surveillance that have been used to ensure fair trading practices. Most models of surveillance depended upon statistical analysis as one of the major tools of data surveillance. But now as trading is moved onto an algorithmic approach, so has the need for defining surveillance in those terms.

2.2 What are the techniques of market manipulation

Traders intent on carrying out fraudulent activities in financial markets rely on a number of methods to profit from the system. Besides the ever constant presence of insider trading, certain techniques have been identified for their unique approach and so we have discussed them in slight detail below. However, since our major focus was on Spoofing, we devoted more space for it further onwards.

2.2.1 Momentum Ignition or Layering

Momentum ignition or Layering is a strategy where a trader initiates a series of orders and trades, in order to cause a rapid price change of the instrument either upwards or downwards and so, induce others to trade at prices which have been artificially altered. The main purpose of this strategy is to create an artificial presence of demand or supply in the market and then make a profit from the resulting movement in price.

2.2.2 Quote Manipulation

Quote manipulation is a strategy usually employed by high-frequency traders(HFTs), who utilize advanced technological communication systems and infrastructure in order to abuse and manipulate the market. This is done in order affect the prices of-of orders placed in dark pools by manipulating prices in the visible markets. Non-bona fide orders are entered on visible marketplaces which change the best bid price and/or the best ask price in order to affect the price calculation at which a trade will occur with a dark order. This activity (which may be combined with abusive liquidity detection) results in a trade with a dark order at an improved price, following which orders are removed from the visible marketplaces.

2.2.3 Spoofing

Spoofing is a fraudulent trading practice where limit orders are placed with an intent to not execute them, in order to manipulate prices. There are various strategies related to the exact execution of this practice, some of which are related to the opening or closing of regular market hours. that involve distorting disseminated market imbalance indicators through the entry of non-bona fide orders, checking for the presence of an iceberg[1] order, affecting a calculated opening price and/or aggressive trading activity near the open or close for an improper purpose.

Spoofing has only fairly recently been defined as an unfair trading practice, and consequently, it is done differently in different markets. One of the first cases of spoofing to be charged involved Navinder Singh Sarao, a British trader, was charged in April 2015 for contributing to the 'Flash Crash' of may 2010.[72].

[1]An iceberg order is a large single order that has been divided into smaller lots, usually through the use of an automated program, for the purpose of hiding the actual order quantity.

2.3 Anomalies

Many times, there exist certain deviations in the trading data, when it is taken as a whole. These deviations from normal trading patterns or behavior might not be illegal presently, but they do count as bending of the rules. When the data is analyzed via such techniques like machine learning, these anomalies can be detected and identified. More about anomaly detection is given under the machine learning section.

2.4 Big data processing

Big data is a well-known term that has been around for two decades in every field and aspect of life. Big data itself is described as a large amount of data with high-variety of information and high-velocity of growth, that it is so complex that it needs a new way to be processed [36]. Batch processing and stream processing is some example of big data processing.

A batch processing is used to process numbers of jobs simultaneously by putting the jobs together in a batch form. The number of jobs in a batch is called the batch size and the maximum value for the batch size is dependent on the machine [26]. In general, batch processing is used to compute large and complex tasks and is mainly focusing on throughput rather than the latency of individual components of the computation. Therefore the latency is measured in minutes or larger units [47]. This project use batch processing with the purpose of making an application that could analyze historical data.

2.5 Software used

This project used several software and tools, as described in this section below.

2.5.1 Spark

Apache Spark is a technology which is providing a fast and general-purpose cluster computing system[62]. It has support for high-level APIs such as Java, Scala, and R. Moreover, it has provided numerous tools for users to use like Spark SQL, MLlib, GraphX and Spark Streaming[62].

2.5.2 Spark MLlib

Spark MLlib is a machine learning library which is provided by Apache Spark with the goal to make machine learning become scalable and easy. Spark MLlib has provided supported tools for machine learning such as[61]:

- Machine learning algorithms: classification, regression, clustering, and collaborative filtering
- Featurization: feature extraction transformation, dimensionality reduction, and selection
- Pipelines: tools for constructing, evaluating, and tuning machine learning pipelines
- Persistence: saving and load algorithms, models, and pipelines
- Utilities: linear algebra, statistics, data handling, etc.

Spark MLlib has support for some programming language such as scala, java, python, and R. In spark ver 2.x, users could use RDD-based or data frames for processing data. However, in the near future, Spark will not support RDD based on processing data. The reason for Spark to be using data frames is because it provides a more user-friendly API than RDD. Spark MLlib has some benefits such as SQL or data from queries, spark data sources, tungsten and catalyst optimizations, and uniform APIs across languages[61]. Moreover, data frames could facilitate practical ML pipelines and feature transformations which are very useful for machine learning. Spark MLlib uses Breeze linear algebra package which depends on netlib-java for optimized numerical processing.

2.5.3 Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) is made to run on commodity hardware and has many similarities with other current distributed file systems but there are important differences such as that HDFS is highly fault-tolerant, provides high throughput and is designed to be deployed on low-cost hardware. HDFS works well with programs using large datasets. [5]

2.5.4 Spring framework

Spring framework is an open source project which provides a stack of technologies and foundational support for different application architecture. It is divided into modules that can be picked at every level of application architecture [53]. We decide to use Spring because its flexibility of configuration without the need of changing source code.

2.5.5 R Programming Language

R is a language and environment for statistical computing and graphics that provides a range of statistical and graphical techniques [15]. It is highly extensible via packages and easy to implement. This project use one of the available packages that are through Comprehensive R Archive Network (CRAN).

2.5.6 Docker

The Docker company that is controlling the container movement and the only container platform provider to address every application across the hybrid cloud [28]. A container itself is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. It is similar to a virtual machine in term of resource isolation and allocation, but functioning differently. Virtual machine virtualizes the hardware, while container virtualizes the operating system hence it's more portable and efficient [27].

2.5.7 Git version control

A system that records changes to a file or several files over time is called version control [18]. Then you can revert back to your older versions of your applications. Git is a version control software that works in these ways [19]:

- Git thinks about the data as a stream of a snapshot. Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot for each time we commit.
- Operation mostly done locally because the project's history is available on local disk.

- Everything in Git is check-summed before it is stored and is then referred to by that checksum.
- Nearly all actions in Git only add data to the Git database and we can experiment without the danger of severely screwing things up.
- There are 3 different states where the files can reside in: committed, modified, and staged. We have a flexibility of which part will be stored.

2.5.8 OpenStack Helion and Horizon

OpenStack is a cloud operating system that controls large pools of computing, storage, and networking resources throughout a datacenter, [13] managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. OpenStack Helion is the newest version of it and we use it through the canonical implementation of OpenStack's Dashboard. It provides a web-based user interface to OpenStack services. [14]

2.5.9 Checkstyle plugin

Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code, makes it ideal for projects that want to enforce a coding standard. Checkstyle is highly configurable and can be made to support almost any coding standard [8]. We made our own configuration with the base of Google Java Style [9] and use the Maven Checkstyle plugin [16] to integrate that into the project.

2.5.10 Circle CI

CircleCI is a modern continuous integration and continuous delivery (CI/CD) platform that automates build, test, and deployment of software [11]. It can be used in the cloud or run it privately on our own server. It runs for every code change in our Github repository to triggers a build and automated tests in their cloud. CircleCI then sends a notification of success or failure after the build and tests complete.

2.5.11 Tableau

Tableau is a data visualization and analytics software with several features, such as interactive dashboard to uncover insight easier, connect to many different data sources, plot the data into a map, share and present the result to others [54]. It is quite easy to use and currently has several types of product for different environments. This project uses Tableau Desktop installed on our workstation.

2.5.12 MATLAB

MATLAB platform is a platform optimized for solving engineering and scientific problems. The matrix-based MATLAB language is the world's most natural way to express computational mathematics with built-in graphics to make it easy to visualize and gain insights from data [39].

3 System architecture

3.1 Hardware architecture

In this project, we have used OpenStack as our cloud computer to build our own server. OpenStack is a cloud operating system which provides storage, network and a large pool of computer resources throughout a datacenter [44]. OpenStack resources can be managed by a dashboard which provides a web interface. We built nine computers in this project, seven of the computers have the disk storage of 220GB, RAM 32 GB, and 8 VCPUs and the other two computers have disk storage of 80 GB, RAM 8GB, and 4 VCPUs.

We used the two computers with lower specifications for two different purposes, one of the computers we used as an ssh-server, this computer had the capability to receive data from SCILA AB company. Later on, this data will be moved to another computer where the data will be processed. The other computer we used to communicate with our local computer and the cluster, we call this computer as proxy-server. The reason we needed this computer was that because a firewall separates our local network from the open stack network. With the proxy-server, we can send our data and program to this computer and we can run our program with Spark and HDFS.

The other seven computers we built with the purpose to run Apache Spark and HDFS. In this project, we used one of them as our master and other computers as workers. The purpose of the master is to control all of the workers and provide a graphical interface where the user can see that status of the HDFS and Apache Spark.

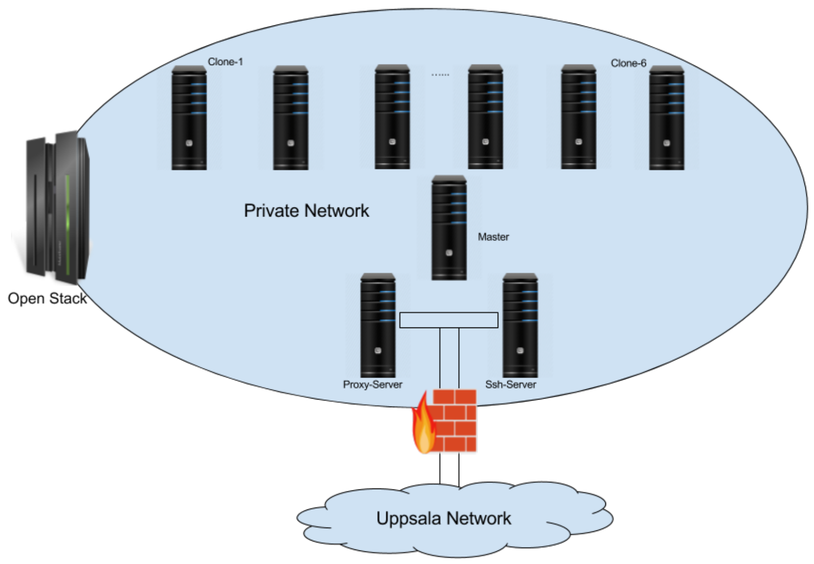


Figure 1: Hardware Architecture

3.2 Software architecture

In this project, we used the two different technologies Apache Spark and Hadoop Distributed File System. We run both of them separately without using Hadoop YARN (Yet Another Resource Negotiator) to unite them.

Apache Spark is used for computing resources and the HDFS is used for our data storage in the cloud. When we are running our program we need to initialize a SparkContext, this allows our code to run parallelized automatically in the cluster[75]. It will help to make the computing process faster than running computing process on a single computer. Moreover, Spark will run in parallel to read or write from the HDFS. We have also made use of several libraries which are provided by Apache Spark, such as Spark SQL and Spark MLLib(machine learning library).

3.3 System overview and operations

This project consists of 3 major components and 1 optional component. The 3 major components are parsing TX files, implementing spoofing detection algorithm, and machine learning utilization. The optional component is data visualization using Tableau or QlikView which is not completed because of the limitation of time and the lack of flexibility to integrated either both option to our prototype. For the optional component, we made a module to generate a report based on specification document. The major components are explained in the next sections.

Other than the mentioned components, there are also several elements that make the whole system running. We use the TX files from Scila as the main data source and then we have optional optimization for saving the parsed data, either store it in parquet files or store it temporarily through caching it. There are several configurations that could be used, depends on which module that we want to run, which is stored in XML files. All elements mentioned is shown in figure 2.

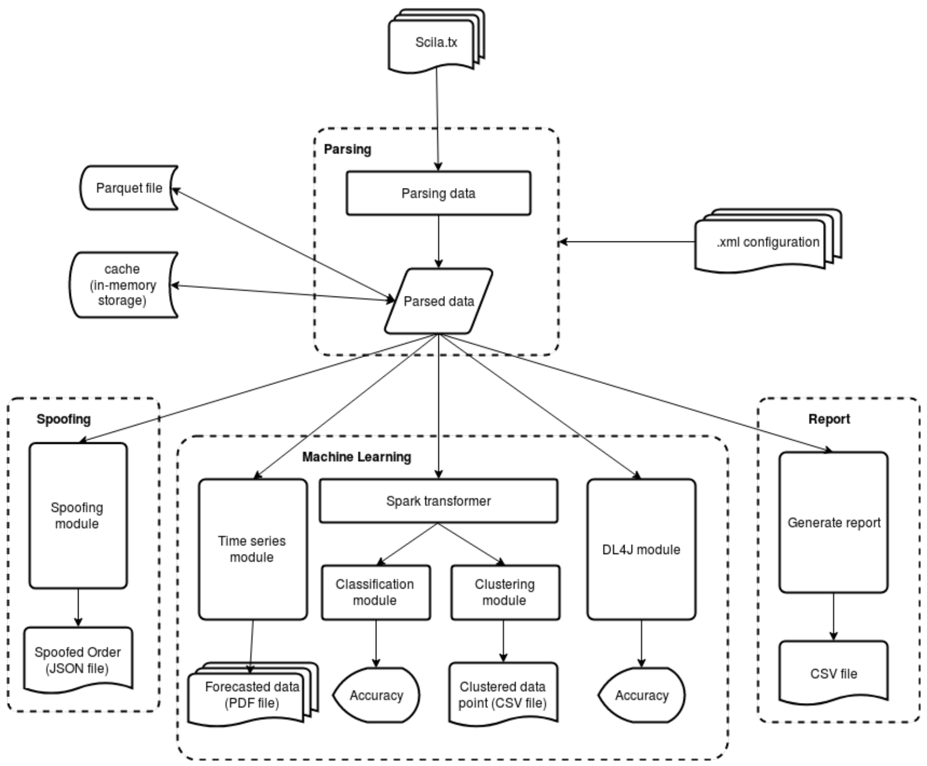


Figure 2: System overview

4 Parsing Implementation

The process of parsing is to analyze a stream of symbols data. The easiest way to understand what parsing data means is to see it as an interpretation of one type of data into some other type of data. The interpretation usually separates and classifies data. Being able to parse is crucial when there is a need to transfer data from program to program. Data will be parsed whenever there is some kind of communication between entities.

In this project it is necessary to read the financial transactional data files used in Scilas software environment into Apache Spark while maintaining the same structure. A key objective is avoiding to pre-process the data files in some stage prior to processing it in Apache Spark. The need for this requirement is due to the project having a criteria of being able to process extremely large quantities of data. Pre-processing the data would demand a majority of the computational power aswell as requiring additional disk storage.

However, it is not possible to import the data files directly without doing any internal processing. All transformation of the data is managed by the Apache Spark Java API.

The process of transforming the raw data consists of the following steps:

1. Unpack Gzip
2. Separate JSON strings
3. Identify what transaction message types each JSON is
4. Rename column names
5. Encode into Java Bean class based dataset

The end goal of the parsing is to create data structures that are usable in later stages of the program, namely to the Spoofing algorithm and the machine learning models. The data is encoded by Apache Spark to improve performance [64].

4.1 Data

```
0000000331{
    "5":["100"],
    "1":"1",
    "7":"Hg",
    "3":"Hg-100",
    "6":1493033323937,
    "10":231,
    "12":1493033323914 }
```

```
{   "2":"100",
    "3":1493033323937,
    "28":730,
    "4":"1000029",
    "5":"SWB",
    "6":"SWB2",
    "7":810000000,
    "8":"O-100-16",
    "9":96200000,
    "10":1493033323937,
    "11":10208,
    "12":true,
    "13":"CANCEL",
    "14":"USER",
    "20":"AAPL USD",
    "30":"/Algo/Alpha1/VWAP/ExecVenueX",
    "31":"" }
```

Figure 3: Example of the raw single line JSON data.

A financial instrument is a virtual or real document that describes a monetary contract between parties [30].

The data is divided into one file for each financial instrument during a day and is structured in a calendar hierarchy of folders with years, months and days. Each file consists of different types of financial transaction data for a certain instrument in a chronological order. Most of the files in our test data are compressed with Gzip and are decompressed during runtime.

Each line in each data file has the structure of two JSON objects with their combined size prepended, as seen in Figure 3. The first JSON object is a header that specifies some internal data and the second JSON objects type of financial transaction message. It is in the second JSON object where data like orders and trade data resides. Apache Spark's JSON reader either supports a single JSON objects over multiple lines or a single line set-up to be able to read it [12].

Therefore these two JSON objects has to separated before applying Apache Spark's JSON reader on the data. Which leads to the need for the program to read each JSON object twice. Once with a custom JSON parser and secondly with the built in Apache Spark reader that converts the JSON into a dataset. To keep track of which header points to which message type, a unique parse ID is inserted into these JSON objects during the separation.

4.1.1 Data Structure

A financial instrument is a virtual or real document that describes a monetary contract between parties [30].

The data is divided into one file for each financial instrument during a day structured in a calendar hierarchy of folders with years, months and days. Each file consists of different types of financial data for a certain instrument in a chronological order. Most of the files in our test data are compressed with Gzip and are decompressed at runtime.

The transformation from the JSON string into a dataset is done in three steps. The specification for the message type contains multiple optional fields in the JSON objects. The data is also filled out to include these optional fields The columns are renamed into their real meaning to improve usability.

The last stage is to provide the schema for the data which allows it to encode it to a strongly typed dataset. To be able to use encoded datasets they need to have a specified schema for their structure that the data provide [64]. Due to this, each message type is filtered out and then encoded into individual datasets for the message types. The schema is defined at runtime with a Java class [64].

4.2 Optimization

The big selling point of Apache Spark and what is advertised mostly is how efficient Apache Spark is with in-memory computations [38]. Even though Apache Spark has a big focus on in-memory computation the system will spill to the disk if the size of the memory is not sufficient, allowing it to run well on any sized data [56].

In contrast to the traditional map-reduce approach Apache Spark tries to do with its in-memory computational pipeline is to minimize writes to disk between dataset transformations [38]. This unlocks the possibility to iteratively perform multiple transformations on a single dataset without the need to write the data before the last stage.

4.2.1 Internal Dataset storage

Since the data files are already divided into a calendar hierarchy there are a number of different possibilities in which order to parse all of the files. Either keeping the same structure or combining some or all levels. The initial implementation of the spoofing algorithm and machine learning models were to process data from multiple days which did not point out the need to parse the data other than in a big chunk.

But since the current spoofing algorithm analyzes the data strictly from a specific day there are two different hierarchy levels configurable for the parsing. One parses all the files in a big chunk and holds each message type in datasets that span across the whole range of the given dates. The other parsing is very similar to how the folder structure for the files look like. It parses each message type into datasets but partition them day-by-day.

The advantage of storing the data in a day-by-day basis is that it is possible to cache each dataset for the spoofing algorithm more efficiently. Allowing the program to cache each day and thereafter delete the dataset from the cache as they are finished being processed. Compared to the other way where all the data is cached before any additional work is done on the data.

As seen in Figure 4 extracting a day's data from the big partitioned datasets struggles a lot when the data is not cached. But as seen in Figure 5 the difference is runtime is massively reduced when Apache Spark can operate on cached data.

As mentioned in 4.1.1 the data is encoded with Java classes into strongly typed datasets to have the possibility to use powerful lambda functions and utilize typed fields [65]. Normally found in the older Resilient Distributed Dataset (RDD) data structure that Apache Spark have had since its initial version [65]. The choice fell on the newer Dataset and DataFrame data structure interface due to improved SQL performance compared to RDD with an optimization engine called Tungsten [20].

Lastly during the parsing there is another optimization that is experimented with to further improve performance of the SQL queries. All transformations in Apache Spark are lazy [67], this means that every time some data is needed in a computation it is delayed as long as possible. This is done to mitigate unnecessary processing of data that is never used. Since there is a lot of preprocessing done with the data to extract what is needed the laziness of Apache Spark is lost during the parsing. All the data has to be read no matter how heavy a query on the data might be.

A workaround is to after the parsing write the parsed data back to the disk and then re-parse it directly to each message type to be able to properly leverage the Apache Spark's lazy evaluation. If the same data is used for multiple executions this can improve both parsing and queries on the data by a huge margin. This can also be combined with a much better data format than JSON. In the program this can be configured to do with the Parquet data format.

4.3 Benchmark

Like previously mentioned there are two different configurable ways that the program can parse the data. But we have also explored another third option. It is very similar to the day-by-day parsing. Instead of parsing all the data beforehand, it keeps the same day-by-day partitioning but incrementally parses one day's data and then to the required queries on it before parsing the rest of the data.

The benchmarks in Figure 4 and 5 ran on a local machine with a 4 cored hyper threaded CPU (total of 8 threads) [29].

Even though caching the datasets speeds up the query it is interesting to see that the query gained speed when using the 4 hyper threaded threads when operating on non cached datasets as seen in Figure 4 compared to in Figure 5 where it only slightly helped or even gave worse performance on the same amount of threads.

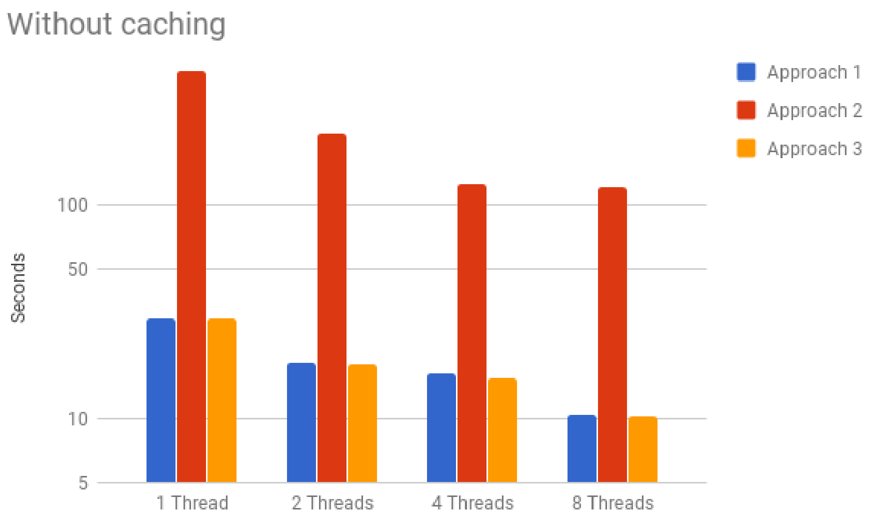


Figure 4: The query *select count(*)* is done on non cached datasets of 1.4 GB in 1300 files covering 15 days of test data. Small partition parsing is denoted as Approach 1. Big partition parsing is denoted as Approach 2. Incremental parsing is denoted as Approach 3.

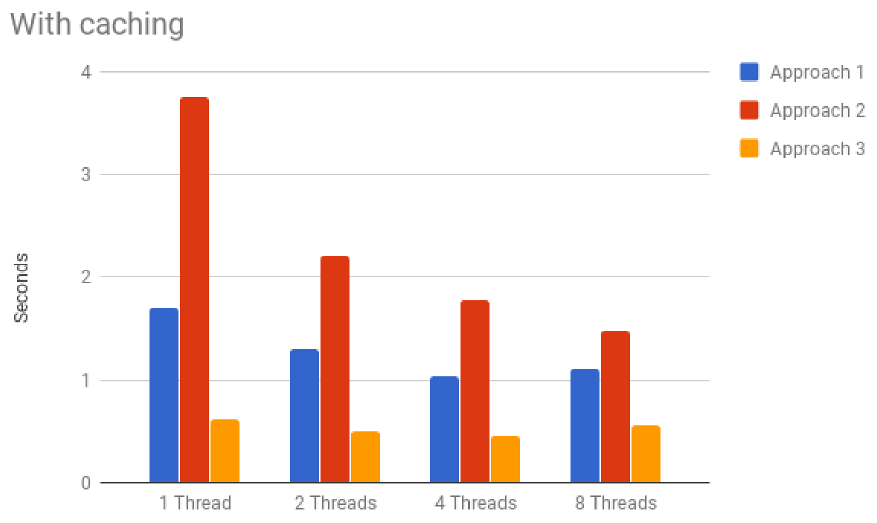


Figure 5: The query *select count(*)* is done on cached datasets of 1.4 GB in 1300 files covering 15 days of test data. Small partition parsing is denoted as Approach 1. Big partition parsing is denoted as Approach 2. Incremental parsing is denoted as Approach 3. Does not include the time taken to cache the dataset

5 Spoofing Detection Implementation

To detect the presence of a suspected spoofing order within the order table, we needed to build a mechanism that would allow us to filter out the legitimate orders. We created multiple filters that use SQL queries to set the different parameters which allow us to shortlist the suspected spoofing orders from within the given data.

5.1 Filters

We use filters to narrow down the potentially spoofed orders from the datasets containing trades and orders, which we get via the parsing system. The first step involves applying a filter to find all the confirmed trades. Confirmed trades are those trades which were executed during continuous trading and of type auto-matched. Auto-matched means that bid side orders and ask side orders are matched continuously into trades by a trade engine, and when the match occurs, the result is known as a trade. Then three separate filters are run on the same dataset so as to finally get a fully filtered dataset.

The filters are limited to query and output results for one day at a time.

5.1.1 Data

Selecting the range of data to filter is done by using different date intervals. The dates are user-specified and can be found in dates.xml.

5.1.2 SQL

All the queries were done using SQL. The SQL queries could be written in either regular SQL or Spark SQL. We found no significant difference in how they were parsed by Apache Spark after testing them, the only benefit was that Spark SQL was easier to use and provided better readability.

5.1.3 Parameters

The parameters we have chosen to implement were the ones suggested in the specification that was provided by Scila AB. We had no real guidelines when it comes to the values to use for our parameters when running our tests. The values we chose were the ones that we found were the most optimal for the provided data. Different parameters used when filtering is also user-specified using Java beans. These values can be modified in spoof.xml.

- **minSpoofValue** The value of an order is volume and price multiplied. These values are then divided by 1,000,000 as otherwise they get too large to process properly. The first iteration of this parameter was to have the minSpoofValue as the minimum value of a spoofing order, meaning orders get filtered if they are below the value of this parameter. This parameter was pre-defined and hard-coded and did not give good results as the prices and volumes vary in different markets.

  For the second iteration, we changed the parameter, making it percentage based. The percentage would then be compared with the calculated difference in value between the average value before an order within a time interval and the actual value of that order. These changes were done to make the filter follow the market prices and order books in which the orders are in.

- **spoofTime** The time, before the trade, that spoofing orders are looked for. This parameter is the first one to be used as it seems to be filtering out a larger part of the input data. This lets the other filters with heavier computations work on smaller subsets of the original data and therefore increasing the performance of the program.

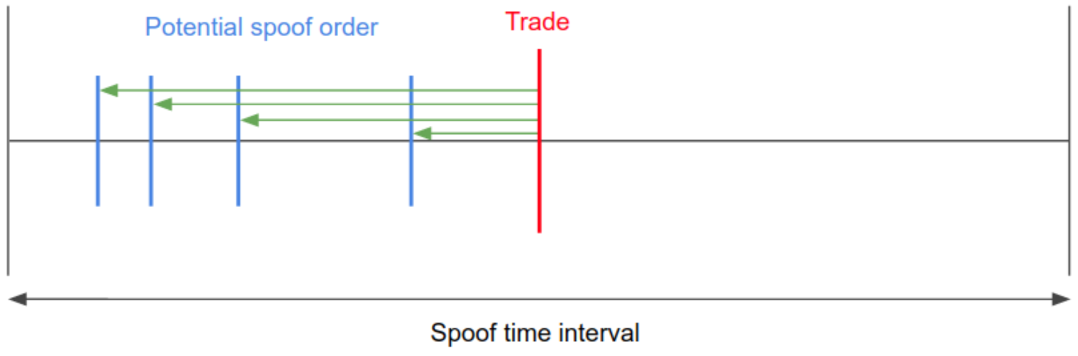


Figure 6: spoofTime usage [**filtering**]

- **participantLevel** The level of the participant. The levels are defined in a hierarchy where ‘member’ is the top level, ‘user’ the second and ‘endUserRef’ the third.

The default value for this parameter is ‘endUserRef’, as this is the most common level where spoofing occurs.

- **spoofCancelPerc** This value is compared with the total amount a user has canceled. The order is kept if the total amount the user has canceled is greater than or equal to this parameter. A user either cancels an order completely or reduces its volume by spoofCancelPerc or above, therefore making an implicit cancel. This happens within the specified spoofTime.
- **minPriceDifference** The minPriceDifference parameter is used to check whether a user made an implicit cancel or an implicit insert. The parameter is the minimum difference in percentage between a current, previous or trade price. An implicit cancel happens if the previous price is near the traded price and the current price is not near the trade price. An implicit insert happens if it was not an implicit cancel and the current price is near the trade price.

5.2 Result and output

The final output is filtered datasets with suspected spoofed orders for each day. These are written into either a JSON or CSV file. The output folder has the same folder structure as the input data. The output folder contains a year folder with months and days in that specific order. Inside every day folder is a file with the date of the specific dataset.

5.2.1 JSON

Example of an alert containing an order suspicious of spoofing. The JSON follows the structure of a scila alert message:

```
[{
"11":"GOOGE595",
"12":"SWB",
"13":"SWB3",
"14":"1000038",
"1":"",
"2":"",
"3":"",
"4":"",
"5":"",
"6":"",
"7":"",
"8":1504502354944,
"9":500,
"10":"112"
}]
```

5.2.2 CSV

An alternative output is creating CSV files containing all suspicious spoofed orders. The reason why this output method was kept is that because it keeps all information about the orders and even our own made columns containing parameter values.

5.3 Benchmark

Very similar to how the parsing performance results as seen in Figure 4 and 5 the spoofing algorithm follows the same structure. The performance with cached datasets are much improved.

Even though the Parquet file format is faster it is not notibly faster. Cached performance is both helping and declining the performance.

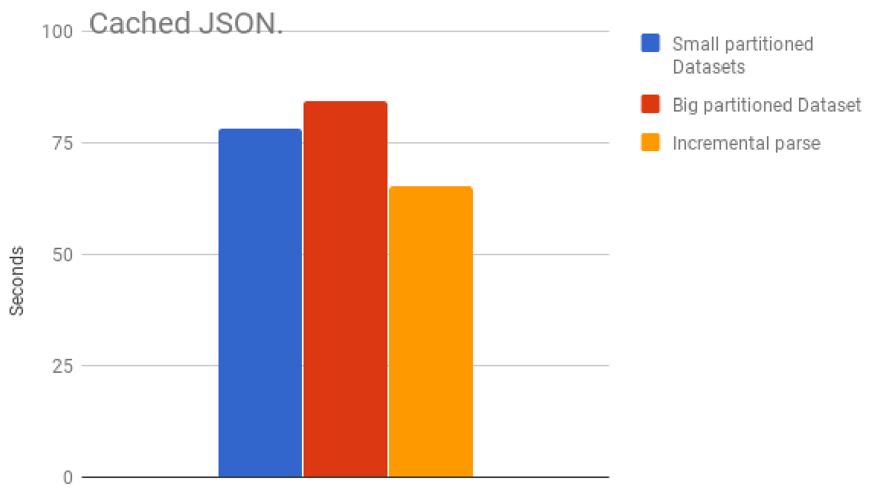


Figure 7: Performance of the spoofing algorithm including writing results to disk from non cached datasets parsed from JSON text files of 1.4 GB in 1300 files covering 15 days of test data. On a 4 core machine (Intel i7 7800) with Spark in local mode.

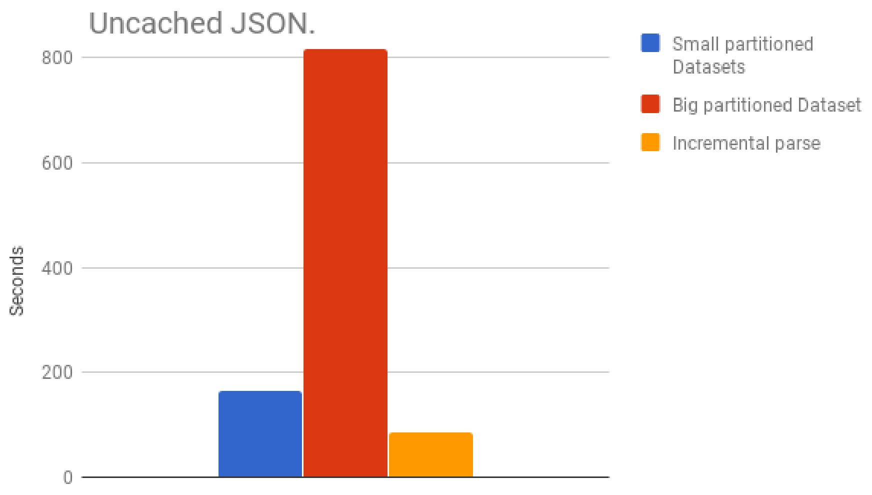


Figure 8: Performance of the spoofing algorithm including writing results to disk from non cached datasets parsed from JSON text files of 1.4 GB in 1300 files covering 15 days of test data. On a 4 core machine (Intel i7 7800) with Spark in local mode.

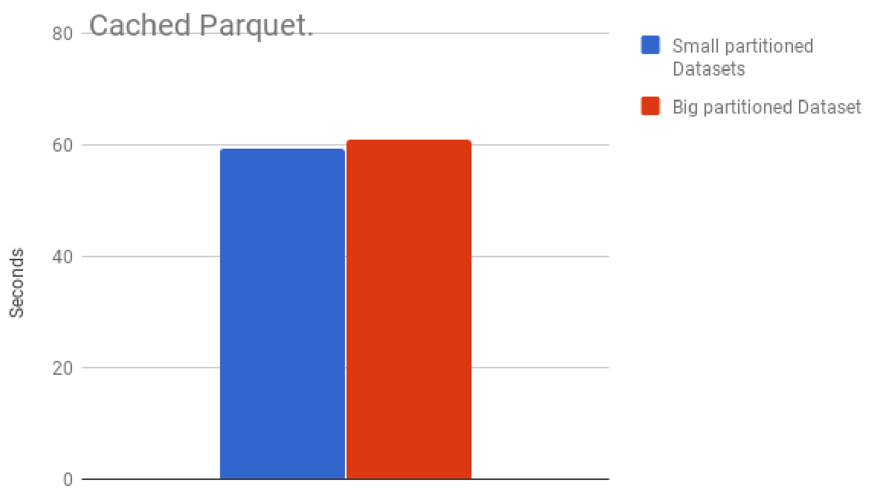


Figure 9: Performance of the spoofing algorithm including writing results to disk from cached datasets parsed from Parquet files of 1.4 GB in 1300 files covering 15 days of test data. On a 4 core machine (Intel i7 7800) with Spark in local mode.

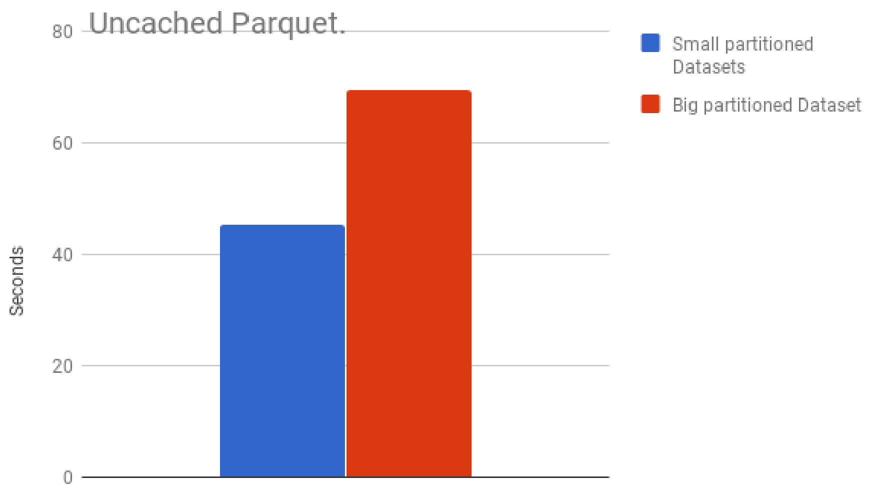


Figure 10: Performance of the spoofing algorithm including writing results to disk from non cached datasets parsed from Parquet files of 1.4 GB in 1300 files covering 15 days of test data. On a 4 core machine (Intel i7 7800) with Spark in local mode.

6 Machine Learning Implementation

The main idea of using machine learning for this project is to detect anomalies in a collection of available parsed datasets, in addition to that we also tried different approaches of utilizing machine learning. We came up with 3 different problems that could be solved using different machine learning techniques within the tools and limitation that we had. Those problems are anomalies detection using unsupervised learning (clustering), classifying market participant based on historical trade data using supervised learning (classification), and forecasting stock closing price using a time series algorithm. Each one of them is explained in this section.

6.1 Data transformation in Spark

Spark has supported many kinds of data transformers for machine learning. We have used some of the features transformers such as String indexer, one-hot encoder, PCA, standard scaler, vector assembler. We have used feature selectors like vector slicer. Parsed data is in the form of Spark Dataframes and it needs to be transformed into a feature vector and a label (optional). The label and selected feature is defined in each approach of machine learning. The common steps that we used to transform the data are shown in the figure.

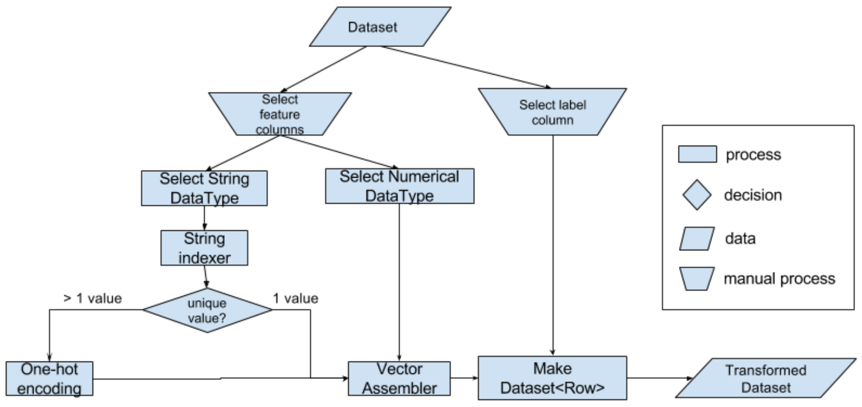
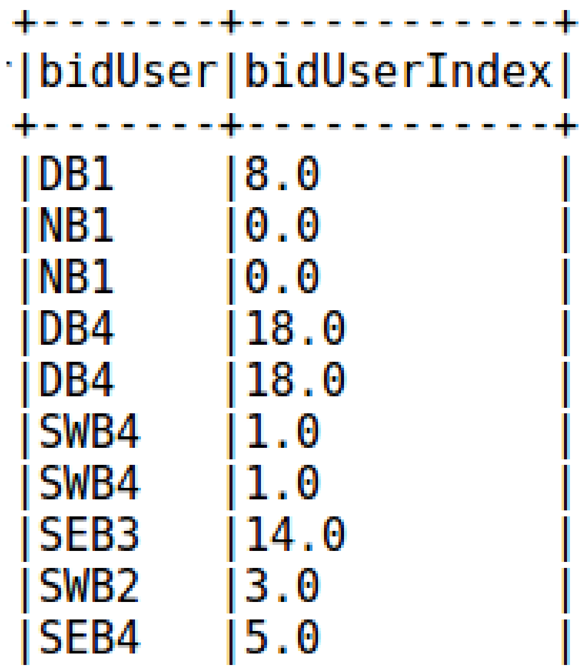


Figure 11: General implementation of spark transformer

6.1.1 StringIndexer

StringIndexer encodes a string column of labels into a column of label indices[66]. The indices are from 0 up to the amount of unique labels, starting from 0 for the most frequent label. The unseen labels will be put at the end of indices. It was used for all categorical values and precedes the process of one-hot encoding.

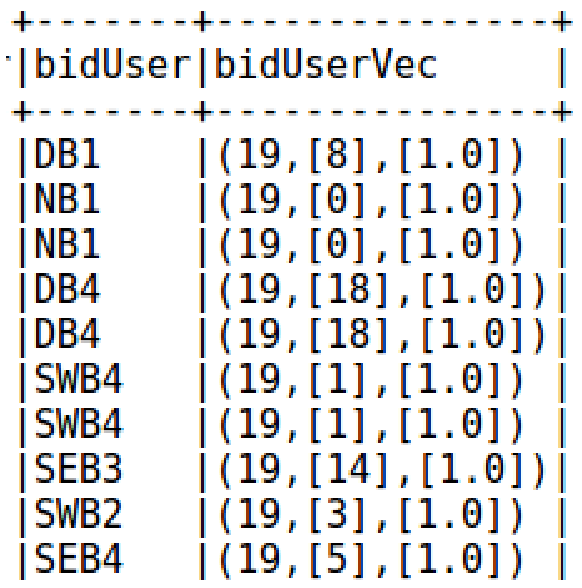


```
+-------+------------+
|bidUser|bidUserIndex|
+-------+------------+
|DB1    |8.0         |
|NB1    |0.0         |
|NB1    |0.0         |
|DB4    |18.0        |
|DB4    |18.0        |
|SWB4   |1.0         |
|SWB4   |1.0         |
|SEB3   |14.0        |
|SWB2   |3.0         |
|SEB4   |5.0         |
```

Figure 12: Result of String Indexer

6.1.2 One-hot Encoding

One-hot Encoding maps a column of label indices to a column of binary vectors, with at most a single one-value[66]. It is suitable for categorical values that do not have the ordinal relationship among them [7]. It was used for all categorical attributes with more than 1 unique value.

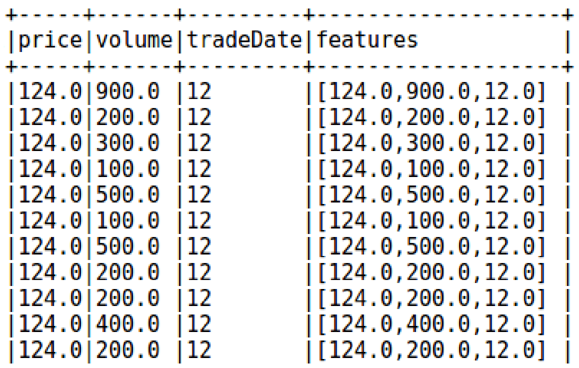


```
+-------+---------------+
|bidUser|bidUserVec     |
+-------+---------------+
|DB1    |(19,[8],[1.0]) |
|NB1    |(19,[0],[1.0]) |
|NB1    |(19,[0],[1.0]) |
|DB4    |(19,[18],[1.0])|
|DB4    |(19,[18],[1.0])|
|SWB4   |(19,[1],[1.0]) |
|SWB4   |(19,[1],[1.0]) |
|SEB3   |(19,[14],[1.0])|
|SWB2   |(19,[3],[1.0]) |
|SEB4   |(19,[5],[1.0]) |
```

Figure 13: Result of One-hot Encoding

6.1.3 VectorAssembler

VectorAssembler is a transformer that combines a given list of columns into a single vector column[66]. It is useful for combining raw features and features generated by different feature transformers into a single feature vector. It accepts all numeric types, boolean types, and vector type. In each row, the values of the input columns will be concatenated into a vector in the specified order.

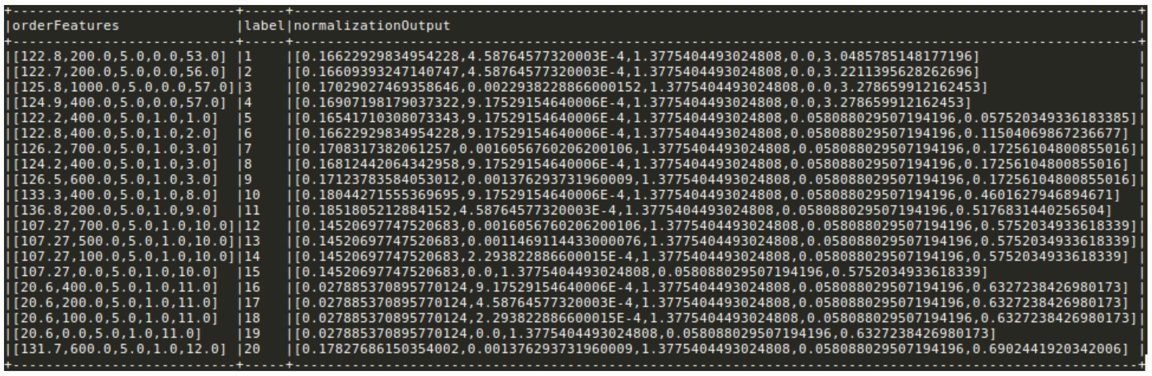


```
+-----+------+---------+------------------+
|price|volume|tradeDate|features          |
+-----+------+---------+------------------+
|124.0|900.0 |12       |[124.0,900.0,12.0]|
|124.0|200.0 |12       |[124.0,200.0,12.0]|
|124.0|300.0 |12       |[124.0,300.0,12.0]|
|124.0|100.0 |12       |[124.0,100.0,12.0]|
|124.0|500.0 |12       |[124.0,500.0,12.0]|
|124.0|100.0 |12       |[124.0,100.0,12.0]|
|124.0|500.0 |12       |[124.0,500.0,12.0]|
|124.0|200.0 |12       |[124.0,200.0,12.0]|
|124.0|200.0 |12       |[124.0,200.0,12.0]|
|124.0|400.0 |12       |[124.0,400.0,12.0]|
|124.0|200.0 |12       |[124.0,200.0,12.0]|
```

Figure 14: Result of VectorAssembler

6.1.4 StandardScaler

StandardScaler is an estimator which can be fit to a dataset of vector rows to produce a dataset to have unit standard deviation and/or zero mean features by computing summary statistics[66]. It has two parameters withStd and withMean. The withStd parameter is set to true by default and it has a function for scales data to units standard deviation. The withMean parameter is set false by defaults and it has a function to build a dense output while users have sparse input data. The result of this method show in 15

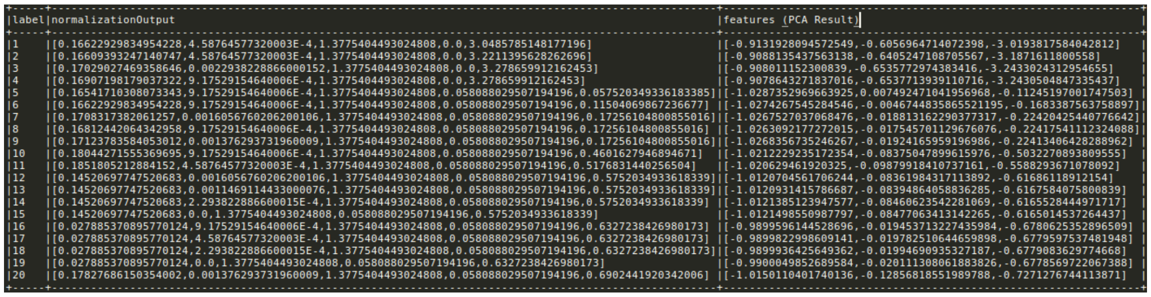


```
+---------------------------+-----+------------------------------------------------------------------------------------------------------+
|orderFeatures              |label|normalizationOutput                                                                                   |
+---------------------------+-----+------------------------------------------------------------------------------------------------------+
|[122.8,200.0,5.0,0.0,53.0] |1    |[0.16622929834954228,4.58764577320003E-4,1.3775404493024808,0.0,3.0485785148177196]                   |
|[122.7,200.0,5.0,0.0,56.0] |2    |[0.16609393247140747,4.58764577320003E-4,1.3775404493024808,0.0,3.2211395628262696]                   |
|[125.8,1000.0,5.0,0.0,57.0]|3    |[0.17029027469358646,0.0022938228866000152,1.3775404493024808,0.0,3.278659912162453]                  |
|[124.9,400.0,5.0,0.0,57.0] |4    |[0.16907198179037322,9.17529154640006E-4,1.3775404493024808,0.0,3.278659912162453]                    |
|[122.2,400.0,5.0,1.0,1.0]  |5    |[0.16541710308073343,9.17529154640006E-4,1.3775404493024808,0.058088029507194196,0.057520349336183385]|
|[122.8,400.0,5.0,1.0,2.0]  |6    |[0.16622929834954228,9.17529154640006E-4,1.3775404493024808,0.058088029507194196,0.11504069867236677] |
|[126.2,700.0,5.0,1.0,3.0]  |7    |[0.1708317382061257,0.0016056760206200106,1.3775404493024808,0.058088029507194196,0.17256104800855016]|
|[124.2,400.0,5.0,1.0,3.0]  |8    |[0.16812442064342958,9.17529154640006E-4,1.3775404493024808,0.058088029507194196,0.17256104800855016] |
|[126.5,600.0,5.0,1.0,3.0]  |9    |[0.17123783584053012,0.001376293731960009,1.3775404493024808,0.058088029507194196,0.17256104800855016]|
|[133.3,400.0,5.0,1.0,8.0]  |10   |[0.18044271555369695,9.17529154640006E-4,1.3775404493024808,0.058088029507194196,0.4601627946894671]  |
|[136.8,200.0,5.0,1.0,9.0]  |11   |[0.1851805212884152,4.58764577320003E-4,1.3775404493024808,0.058088029507194196,0.5176831440256504]   |
|[107.27,700.0,5.0,1.0,10.0]|12   |[0.14520697747520683,0.0016056760206200106,1.3775404493024808,0.058088029507194196,0.5752034933618339]|
|[107.27,500.0,5.0,1.0,10.0]|13   |[0.14520697747520683,0.0011469114433000076,1.3775404493024808,0.058088029507194196,0.5752034933618339]|
|[107.27,100.0,5.0,1.0,10.0]|14   |[0.14520697747520683,2.293822886600015E-4,1.3775404493024808,0.058088029507194196,0.5752034933618339] |
|[107.27,0.0,5.0,1.0,10.0]  |15   |[0.14520697747520683,0.0,1.3775404493024808,0.058088029507194196,0.5752034933618339]                  |
|[20.6,400.0,5.0,1.0,11.0]  |16   |[0.027885370895770124,9.17529154640006E-4,1.3775404493024808,0.058088029507194196,0.6327238426980173] |
|[20.6,200.0,5.0,1.0,11.0]  |17   |[0.027885370895770124,4.58764577320003E-4,1.3775404493024808,0.058088029507194196,0.6327238426980173] |
|[20.6,100.0,5.0,1.0,11.0]  |18   |[0.027885370895770124,2.293822886600015E-4,1.3775404493024808,0.058088029507194196,0.6327238426980173]|
|[20.6,0.0,5.0,1.0,11.0]    |19   |[0.027885370895770124,0.0,1.3775404493024808,0.058088029507194196,0.6327238426980173]                 |
|[131.7,600.0,5.0,1.0,12.0] |20   |[0.17827686150354002,0.001376293731960009,1.3775404493024808,0.058088029507194196,0.6902441920342006] |
+---------------------------+-----+------------------------------------------------------------------------------------------------------+
```

Figure 15: Result of Standard Scaler

6.1.5 Principal Component Analysis

Principal component analysis(PCA) is a statistical method that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of the linearly uncorrelated variable called principal component[66]. The main function of PCA is to project a vector from high dimensionality to lower dimensionality. In our project, we have done a projection of 5 dimensional to 3 dimensional of principal components. The result of this method shown in figure 16.



```
+-----+-----------------------------------------------------------------------------------------------------+------------------------------------------------------------------------+
|label|normalizationOutput                                                                                  |features (PCA Result)                                                   |
+-----+-----------------------------------------------------------------------------------------------------+------------------------------------------------------------------------+
|1    |[0.16622929834954228,4.58764577320003E-4,1.3775404493024808,0.0,3.0485785148177196]                  |[-0.9131928094572549,-0.6056964714072398,-3.0193817584042812]           |
|2    |[0.16609393247140747,4.58764577320003E-4,1.3775404493024808,0.0,3.2211395628262696]                  |[-0.9088135437563138,-0.6405247108705567,-3.1871611800558]              |
|3    |[0.17029027469358646,0.0022938228866000152,1.3775404493024808,0.0,3.278659912162453]                 |[-0.908011152300839,-0.6535772974383416,-3.2433024312954655]            |
|4    |[0.16907198179037322,9.17529154640006E-4,1.3775404493024808,0.0,3.278659912162453]                   |[-0.9078643271837016,-0.6537713939110716,-3.2430504847335437]           |
|5    |[0.16541710308073343,9.17529154640006E-4,1.3775404493024808,0.058088029507194196,0.057520349336183385]|[-1.0287352969663925,0.007492471041956968,-0.11245197001747503]         |
|6    |[0.16622929834954228,9.17529154640006E-4,1.3775404493024808,0.058088029507194196,0.11504069867236677] |[-1.0274267545284546,-0.004674483586552119 5,-0.1683387563758897]       |
|7    |[0.1708317382061257,0.0016056760206200106,1.3775404493024808,0.058088029507194196,0.17256104800855016]|[-1.0267527037068476,-0.018813162290377317,-0.22420425440776642]        |
|8    |[0.16812442064342958,9.17529154640006E-4,1.3775404493024808,0.058088029507194196,0.17256104800855016] |[-1.0263092177272015,-0.017545701129676076,-0.22417541112324088]        |
|9    |[0.17123783584053012,0.0013762937319600009,1.3775404493024808,0.058088029507194196,0.17256104800855016]|[-1.0268356735246267,-0.01924165959196986,-0.22413406428288962]        |
|10   |[0.18044271555369695,9.17529154640006E-4,1.3775404493024808,0.058088029507194196,0.4601627946894671]  |[-1.0212229235172354,-0.08375047899615976,-0.5032270893809555]          |
|11   |[0.1851805212884152,4.58764577320003E-4,1.3775404493024808,0.058088029507194196,0.5176831440256504]   |[-1.020629461920325,-0.09879918410737161,-0.5588293671078092]           |
|12   |[0.14520697747520683,0.0016056760206200106,1.3775404493024808,0.058088029507194196,0.5752034933618339]|[-1.0120704561706244,-0.08361984317113892,-0.61686118912154]            |
|13   |[0.14520697747520683,0.0011469114433000076,1.3775404493024808,0.058088029507194196,0.5752034933618339]|[-1.0120931415786687,-0.08394864058836285,-0.6167584075800839]          |
|14   |[0.14520697747520683,2.293822886600015E-4,1.3775404493024808,0.058088029507194196,0.5752034933618339] |[-1.0121385123947577,-0.08460623542281069,-0.6165528444971717]          |
|15   |[0.14520697747520683,0.0,1.3775404493024808,0.058088029507194196,0.5752034933618339]                 |[-1.0121498550987797,-0.08477063413142265,-0.6165014537264437]          |
|16   |[0.027885370895770124,9.17529154640006E-4,1.3775404493024808,0.058088029507194196,0.6327238426980173] |[-0.9899596144528696,-0.01945371322743598 4,-0.6780625352896509]        |
|17   |[0.027885370895770124,4.58764577320003E-4,1.3775404493024808,0.058088029507194196,0.6327238426980173] |[-0.9899822998609141,-0.019782510644659898,-0.6779597537481948]         |
|18   |[0.027885370895770124,2.293822886600015E-4,1.3775404493024808,0.058088029507194196,0.6327238426980173]|[-0.9899936425649362,-0.019946909353271870,-0.6779083629774668]         |
|19   |[0.027885370895770124,0.0,1.3775404493024808,0.058088029507194196,0.6327238426980173]                |[-0.9900049852689584,-0.020111308061883826,-0.6778569722067388]         |
|20   |[0.17827686150354002,0.0013762937319600009,1.3775404493024808,0.058088029507194196,0.6902441920342006] |[-1.0150110401740136,-0.12856818551989788,-0.7271276744113871]          |
+-----+-----------------------------------------------------------------------------------------------------+------------------------------------------------------------------------+
```

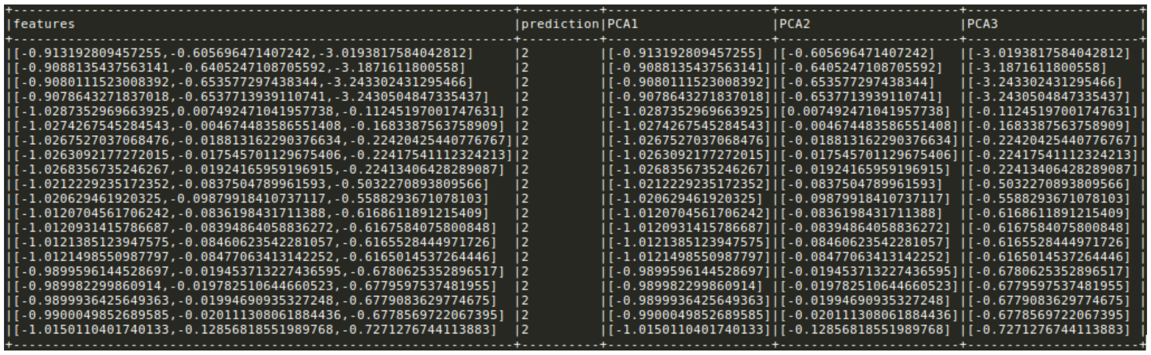
Figure 16: Result of PCA

6.1.6 VectorSlicer

VectorSlicer is a transformer function that takes a feature vector and outputs a new feature with a sub-array of the original features[66]. Vector slicer accepts a vector with indices which specified by users. The outputs of vector slicer will be a new vector column with values from specified indices. Vector slicer could accept two types of input. The two types of inputs are

- Integer indices which represent the number of index vectors which we want to retrieve, setIndices().
- String indices which represent the names of features into the vector, setNames(). This type requires vector column have an attribute group. It will result in order features with the order given by users while they choose which sub-array of a vector to choose.

The result of this method can be seen in figure 17.



```
+------------------------------------------------------------+----------+---------------------+----------------------+---------------------+
|features                                                    |prediction|PCA1                 |PCA2                  |PCA3                 |
+------------------------------------------------------------+----------+---------------------+----------------------+---------------------+
|[-0.913192809457255,-0.605696471407242,-3.0193817584042812] |2         |[-0.913192809457255] |[-0.605696471407242]  |[-3.0193817584042812] |
|[-0.9088135437563141,-0.6405247108705592,-3.1871611800558]  |2         |[-0.9088135437563141]|[-0.6405247108705592] |[-3.1871611800558]    |
|[-0.9080111523008392,-0.653577297438344,-3.243302431295466] |2         |[-0.9080111523008392]|[-0.653577297438344]  |[-3.243302431295466]  |
|[-0.9078643271837018,-0.6537713939110741,-3.2430504847335437]|2        |[-0.9078643271837018]|[-0.6537713939110741] |[-3.2430504847335437] |
|[-1.0287352969663925,0.007492471041957738,-0.11245197001747631]|2      |[-1.0287352969663925]|[0.007492471041957738]|[-0.11245197001747631]|
|[-1.0274267545284543,-0.004674483586551408,-0.1683387563758909]|2      |[-1.0274267545284543]|[-0.004674483586551408]|[-0.1683387563758909]|
|[-1.0267527037068476,-0.018813162290376634,-0.22420425440776767]|2     |[-1.0267527037068476]|[-0.018813162290376634]|[-0.22420425440776767]|
|[-1.0263092177272015,-0.017545701129675406,-0.22417541112324213]|2     |[-1.0263092177272015]|[-0.017545701129675406]|[-0.22417541112324213]|
|[-1.0268356735246267,-0.019241659591969 15,-0.22413406428289087]|2     |[-1.0268356735246267]|[-0.01924165959196915] |[-0.22413406428289087]|
|[-1.0212229235172352,-0.0837504789961593,-0.5032270893809566]|2        |[-1.0212229235172352]|[-0.0837504789961593] |[-0.5032270893809566] |
|[-1.020629461920325,-0.09879918410737117,-0.5588293671078103]|2        |[-1.020629461920325] |[-0.09879918410737117]|[-0.5588293671078103] |
|[-1.0120704561706242,-0.0836198431711388,-0.6168611891215409]|2        |[-1.0120704561706242]|[-0.0836198431711388] |[-0.6168611891215409] |
|[-1.0120931415786687,-0.08394864058836272,-0.6167584075800848]|2       |[-1.0120931415786687]|[-0.08394864058836272]|[-0.6167584075800848] |
|[-1.0121385123947575,-0.08460623542281057,-0.6165528444971726]|2       |[-1.0121385123947575]|[-0.08460623542281057]|[-0.6165528444971726] |
|[-1.0121498550987797,-0.08477063413142252,-0.6165014537264446]|2       |[-1.0121498550987797]|[-0.08477063413142252]|[-0.6165014537264446] |
|[-0.9899596144528697,-0.019453713227436595,-0.6780625352896517]|2      |[-0.9899596144528697]|[-0.019453713227436595]|[-0.6780625352896517]|
|[-0.989982299860914,-0.019782510644660523,-0.6779597537481955]|2       |[-0.989982299860914] |[-0.019782510644660523]|[-0.6779597537481955]|
|[-0.9899936425649363,-0.01994690935327248,-0.6779083629774675]|2       |[-0.9899936425649363]|[-0.01994690935327248]|[-0.6779083629774675] |
|[-0.9900049852689585,-0.020111308061884436,-0.6778569722067395]|2      |[-0.9900049852689585]|[-0.020111308061884436]|[-0.6778569722067395]|
|[-1.0150110401740133,-0.12856818551989768,-0.7271276744113883]|2       |[-1.0150110401740133]|[-0.12856818551989768]|[-0.7271276744113883] |
+------------------------------------------------------------+----------+---------------------+----------------------+---------------------+
```

Figure 17: Result of Vector Slicer

6.1.7 Normalizer

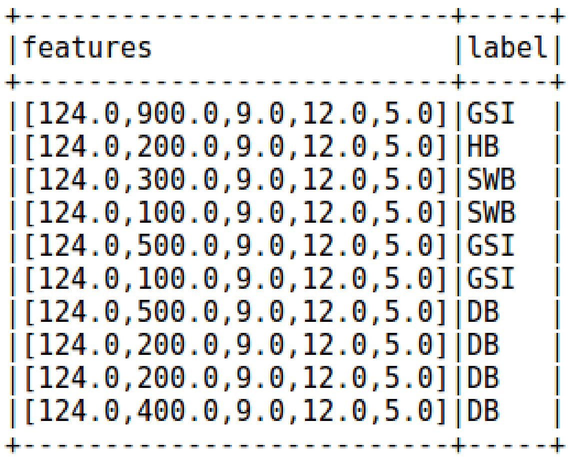
Normalizer transforms a dataset of Vector rows by normalizing each Vector to have unit norm [66]. It uses $p - norm$ with a default value of 2 which could be changed into an integer with value more than 1 until infinity. This parameter will define what kind of normalization that will be used. There are Manhattan norm ($p - norm = 1$), Euclidean norm ($p - norm = 2$), and infinity norm ($p - norm > 2$). Figure 19 shows the result of normalized vector using $p - norm = 2$. This normalization can help standardize your input data and improve the behavior of learning algorithms.

6.2 Classifying market participant in Trade Dataset

6.2.1 Introduction to classification

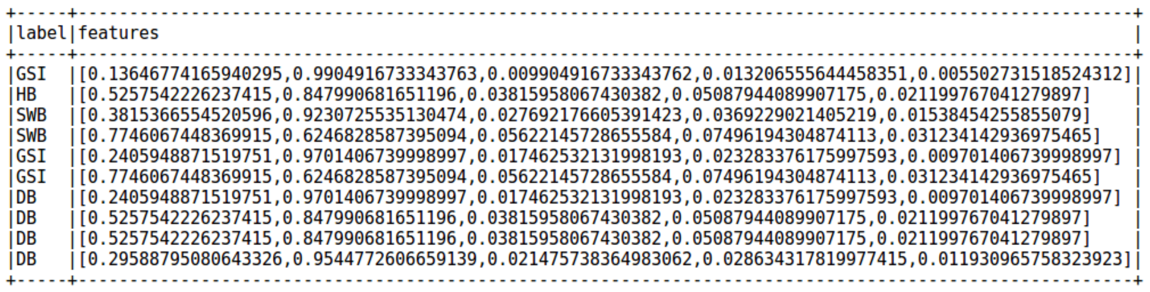
Classification is one sub-category in supervised learning, where the purpose is identifying the class of a new instance based on a training set of data containing instances whose class membership is known and the label is a discrete value [74] [71]. This project uses classification method in order to determine which trade belongs to whom based on selected attributes.

This is one of the approaches that we did to explore what we could do with Spark MLlib and it was suggested by Scila. This particular scenario is a multi-class classification problem based on available data. There are 38 unique labels for the end user, 7 unique labels for the member, and 19 unique labels for the user, either when participant side is taken into consideration or not. The dataset will look like figure 18 with the label as the target value (market participant level) and features column is the selected attribute, both is defined by user in ml-beans.xml. Before the dataset being used by the classifier, it was normalized first as seen in figure 19



```
+--------------------------+-----+
|features                  |label|
+--------------------------+-----+
|[124.0,900.0,9.0,12.0,5.0]|GSI  |
|[124.0,200.0,9.0,12.0,5.0]|HB   |
|[124.0,300.0,9.0,12.0,5.0]|SWB  |
|[124.0,100.0,9.0,12.0,5.0]|SWB  |
|[124.0,500.0,9.0,12.0,5.0]|GSI  |
|[124.0,100.0,9.0,12.0,5.0]|GSI  |
|[124.0,500.0,9.0,12.0,5.0]|DB   |
|[124.0,200.0,9.0,12.0,5.0]|DB   |
|[124.0,200.0,9.0,12.0,5.0]|DB   |
|[124.0,400.0,9.0,12.0,5.0]|DB   |
+--------------------------+-----+
```

Figure 18: Transformed dataset



```
+-----+-------------------------------------------------------------------------------------------------------+
|label|features                                                                                               |
+-----+-------------------------------------------------------------------------------------------------------+
|GSI  |[0.13646774165940295,0.9904916733343763,0.009904916733343762,0.013206555644458351,0.005502731518524312]|
|HB   |[0.5257542226237415,0.847990681651196,0.03815958067430382,0.05087944089907175,0.021199767041279897]    |
|SWB  |[0.3815366554520596,0.9230725535130474,0.027692176605391423,0.0369229021405219,0.01538454255855079]    |
|SWB  |[0.7746067448369915,0.6246828587395094,0.05622145728655584,0.07496194304874113,0.031234142936975465]   |
|GSI  |[0.2405948871519751,0.9701406739998997,0.017462532131998193,0.023283376175997593,0.009701406739998997] |
|GSI  |[0.7746067448369915,0.6246828587395094,0.05622145728655584,0.07496194304874113,0.031234142936975465]   |
|DB   |[0.2405948871519751,0.9701406739998997,0.017462532131998193,0.023283376175997593,0.009701406739998997] |
|DB   |[0.5257542226237415,0.847990681651196,0.03815958067430382,0.05087944089907175,0.021199767041279897]    |
|DB   |[0.5257542226237415,0.847990681651196,0.03815958067430382,0.05087944089907175,0.021199767041279897]    |
|DB   |[0.29588795080643326,0.9544772606659139,0.021475738364983062,0.028634317819977415,0.011930965758323923]|
+-----+-------------------------------------------------------------------------------------------------------+
```

Figure 19: Normalized features vector

6.2.2 Classifier

There are 4 different classifiers that we used in this project which are logistic regression, multilayer perceptron, linear support vector machine combined with one-versus-rest, and random forest. Those classifiers were chosen based on their availability and characteristic in Spark MLlib. Each of them has their own characteristic and was easily implemented using available Java API.

- Logistic regression classifier is a statistical model that uses probability to predict a binary outcome. It could also be used for multi-class classification by using multinomial logistic regression. Spark MLlib provide the multinomial logistic regression by implementing softmax function with equation 1 to measure the probability of the outcome classes $k \in 1, 2, ..., K$ [57]. The equation below is a derivation from binary logistic regression as a log-linear model, where X is features and β is regression coefficients corresponding to its outcome [73].

$$P(Y = k|X, \beta_k, \beta_{0k}) = \frac{e^{\beta_k \cdot X + \beta_{0k}}}{\sum_{k'=0}^{K-1} e^{\beta_{k'} \cdot X + \beta_{0k'}}} \tag{1}$$

Logistic regression is usually fast to converge and Spark MLlib uses multinomial response model with the elastic-net penalty to control that it is not overfitting. The algorithm makes sure that all the data points will belong to one of the classes because it uses probability to determine their class.

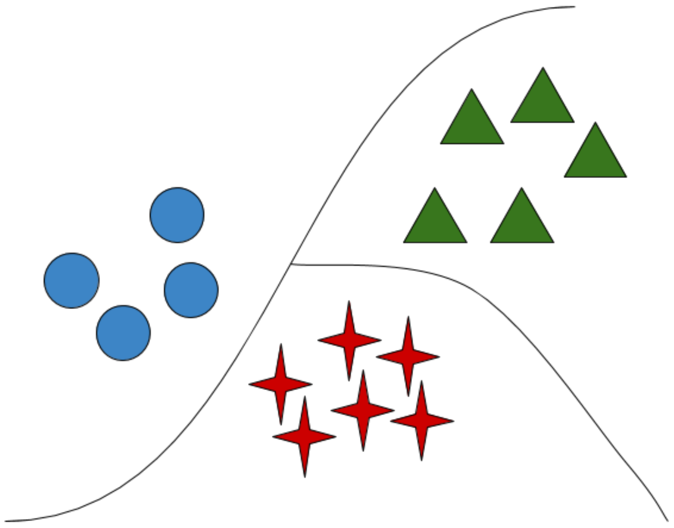


Figure 20: Logistic Regression

- Multilayer perceptron (MLP) is one of the artificial neural network architectures that has more than one single layer. It is a network inspired by the biological neural network. It comprises different layers with different amount of nodes in it 21. Those layers are called input layer, hidden layer, and output layer. The input layer is a layer which receives input from outside the network and the number of nodes is based on the amount of element in features vector. The output layer is a layer which passes along the result from within network to outside and the number of nodes is based on the number of available class. The hidden layer consists of one or more layers fully connected among themselves, input layer, and output layer. Based on this article [48], the number of nodes in hidden layer is equal to sum of number of inputs and outputs multiplied by 2 and divided by 3.

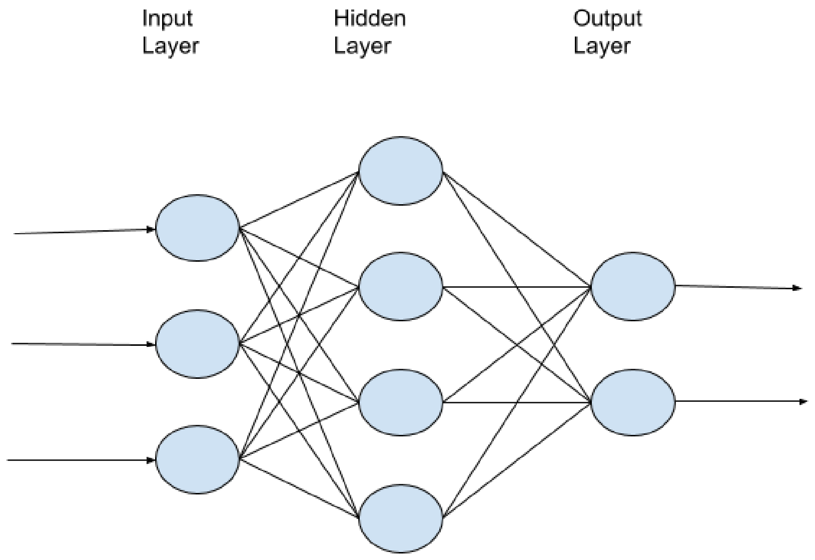


Figure 21: Multi-Layer Perceptron

In general, an artificial neural network has common properties of parallelism, able to generalize within a limit, adaptive to be trained again, and is fault-tolerance. Spark MLlib made this neural network based on feed-forward artificial neural network and employs back-propagation for learning the model. The nodes in hidden layer use sigmoid (logistic) function and the nodes in output layer use softmax function for the activation function.

- Linear Support Vector Classifier (LSVC) and One-versus-Rest are combined to make a Support Vector Machine (SVM) that could handle a multiclass problem. Spark only provided LSVM which could only work for binary classification, but it could be combined with One-versus-Rest. It means that each time the classifier learn, it would take one class and classify that against the rest of the classes as visualized in figure 22. The blue lines are a hyperplane made from the highest margin between support vector from different classes. When those LSVC are combined, there is an undecided area which doesn't belong to any class. The common optimization for this is by using continuous values of SVM decision functions [1], so whichever class has a decision function with the highest value is the class for that data point. It is represented by the black thick line that separate those 3 classes in the figure 22 below.

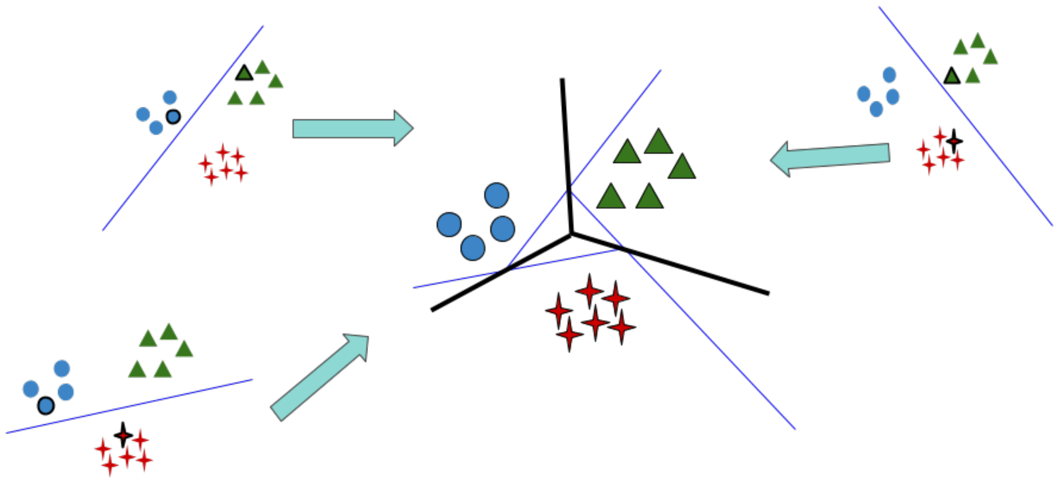


Figure 22: Linear Support Vector Machine combined with One-versus-Rest

- Random Forest classifier works by randomly sampling subsets of the training dataset, fitting a model to this subset, and aggregating the predictions from each tree [60]. It combines many decision trees in order to reduce the risk of overfitting and injects randomness into the training process so that each decision tree is a bit different. The randomness was injected through subsampling the original dataset on each iteration to get a different training set (bootstrapping) and then different random subset of features in each tree node. Prediction for a new instance is through majority vote where each tree's prediction (represented by a thick-bordered circle) is counted as a vote for one class and the label is the class that has the most votes.

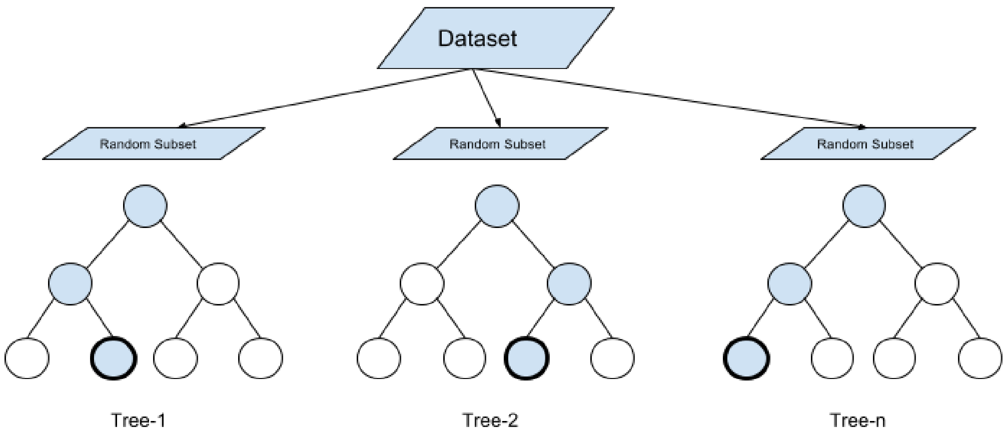


Figure 23: Random Forest

Random forests handle categorical features, extends to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions.

6.2.3 Classification workflow

The workflow that was implemented in this project is shown in the figure 24. The dataset already split into a training dataset and a testing dataset by ratio 7:3 respectively. The training dataset is the dataset used when training the classifier to understand the pattern in the data. The testing dataset is a dataset used for testing the trained model. The general flow would be:

- Build classifier based on user selection.
- Do hyper-parameter tuning if stated true or directly train the classifier, this step uses the training dataset.
- Use the trained model to predict the testing dataset.

- Evaluate the model performance using multiclass metric.

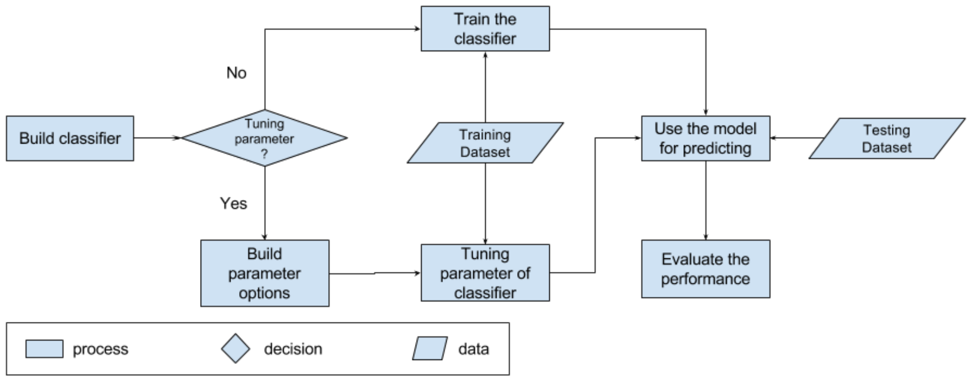


Figure 24: Classification workflow

Hyper-parameter tuning is a process of achieving the best trained model by adjusting the value for each parameter of the classifier. A combination of parameter's value will run the whole classification workflow inside the hyper-parameter tuning process and it iterates for as much as available different combinations. It has 3 part in Spark MLlib:

- $Estimator$ is a pipeline or algorithm to tune.
- $ParamMap$ is sets of parameter combination which could be called as parameter grid.
- $Evaluator$ is a metric to measure the performance of trained model.

Each classifier has different parameter that could be tuned, thus result in different set of parameters for them. Logistic Regression has parameter of regularization, maximum number of iteration, elastic net, boolean condition of fit an intercept term, boolean condition of training data standardization, and depth for treeAggregate of number of partitions in Spark. Linear Support Vector Classifier has the same parameters as Logistic Regression except elastic net. Random Forest has parameters of maximum depth of a tree, number of tree, random seed for bootstraping and choosing subsets, and minimum information gain. Multilayer Perceptron only has parameters of random seed and maximum number of iteration.

Multiclass metric is an evaluator performance of a model that was used for any multiclass problem which entail a measurement for each class separately. The components that are included in this metric is described below.

- **Confusion Matrix** is an error matrix that shows performance of the algorithm for each class.
- **Accuracy** is a percentage of how close to the true value (true value is 100) in classifying the data for all class. It could also be defined as number of true positives divided by sum of true positives and false positives of all class.
- **Precision by label** is a number of true positives divided by the total number of elements (sum of true positive and false positive) labeled as belonging to that particular class. It range from 1 as the best score and 0 as the worst.
- **Recall by label** is a number of true positives divided by the total number of elements that actually belong to the positive class. It measures how many correct prediction for that particular class started from 0 as the lowest and 1 as the highest.
- **F-measure by label** is a measurement that consider precision and recall through their average with 0 as the lowest and 1 as the highest possible result.
- **Weighted precision** is average precision of all classes.
- **Weighted Recall** is average recall of all classes.
- **Weighted F-measure** is average F-measure of all classes.

6.2.4 Result

There were a lot of experiment done with different parameter and configuration when trying to solve this classification problem. We tried to see the difference effect for each different parameter such as different model of classifier, difference between normalizing attributes or not, different type of market participant level, different size of dataset, different attributes, and the effect of hyper-parameter tuning. There are attributes that was extracted from another attribute, for example timeOfTrade is converted into tradeYear, tradeMonth, tradeDate, tradeHour, tradeMinute, and tradeSecond.

The complete list of available attribute to be used is in $ml - columns.xml$. All the features used were normalized except if it's stated otherwise as shown in table 1 where the normalization did not improve the performance significantly. Even though normalization does not give the improvement we are looking for, we still use normalization for all the experiments stated in this section.

Table 1: Accuracy of normalized attributes and raw attributes

Normalized	Logistic Regression	SVM	Random Forest	MLP
No	32.02%	23.29%	33.27%	32.23%
Yes	31.96%	27.25%	32.62%	32.29%

There are 4 different classifier that we used as explained in the beginning. We run them using the dataset of 1 day with attributes of price, volume, tradeHour, tradeMinute, tradeSecond and class of bidMember, except if it is stated otherwise in the table. Multiclass metric gave the result as shown in table 2 and it seems that all the classifiers gave similar result.

Table 2: The result of different classifier

Classifier	Accuracy	Weighted Precision	Weighted Recall	Weighted F-measure
Logistic Regression	31.96%	0.102156	0.319618	0.154826
SVM	27.25%	0.163225	0.27251	0.182492
Random Forest	32.62%	0.369401	0.326178	0.181582
MLP	32.29%	0.247773	0.322898	0.181582

We also tried increasing the dataset size by using a different number of days, from 1 day, 1 month and the whole data. Table 3 shows that the amount of dataset does not really change the model's performance. We also compared the difference between using hyper-parameter tuning or not. Table 4 shown that hyper-parameter tuning is not improving the performance for this dataset.

Table 3: Accuracy of different size of dataset

Date range	Instances	Logistic Regression	SVM	Random Forest	MLP
1 day	11301	31.96%	27.25%	32.62%	32.29%
1 month	134376	31.44%	27.85%	31.45%	31.44%
Whole data	1195040	31.63%	29.84%	31.63%	31.63%

Table 4: Accuracy of hyper-parameter tuning

Hyper-parameter tuning	Logistic Regression	SVM	Random Forest	MLP
No	31.96%	27.25%	32.62%	32.29%
Yes	31.96%	31.99%	31.96%	32.05%

Other experiments was done by looking at different type of market participant as the class as shown in table 5. It shows that the best performance is when we tried using the Member level of market participant. The pattern is more distinguishable if we classify them as Member without the need to look at certain side of participant.

Table 5: Accuracy of different market participant level

Class	Unique labels	Logistic Regression	SVM	Random Forest	MLP
askEndUserRef	38	3.88%	3.34%	9.57%	6.02%
askUser	19	9.99%	5.99%	12.88%	10.58%
askMember	7	30.98%	24.03%	31.45%	30.98%
bidEndUserRef	38	3.46%	3.31%	7.07%	6.56%
bidUser	19	7.57%	6.23%	11.15%	10.17%
bidMember	7	31.96%	27.25%	32.62%	32.29%
allEndUserRef	38	3.19%	3.19%	7.10%	3.85%
allUser	19	9.53%	5.85%	11.04%	9.31%
allMember	7	31.12%	24.06%	31.11%	31.15%

The last experiments that we did was changing the combination of attributes as shown in table 6 where we use 1 month dataset instead of only 1 day. It shows that if we put other level of market participant, the model perform perfectly up until it is too perfect (100% accuracy). It seems that the pattern between market participant level is very distinguishable and thus affect the result. It is also possible that the result in table 6 would be overfitting which then only applies in this particular data that we have.

Table 6: Accuracy of different attribute combination

Attributes	Vector length	Logistic Regression	SVM	Random Forest	MLP
price, volume	2	31.41%	22.94%	31.41%	31.41%
price, volume, tradeDate, tradeHour, tradeMinute, tradeSecond	6	31.55%	24.25%	31.57%	31.55%
price, volume, tradeDate, tradeHour, tradeMinute, tradeSecond, bidUser	25	99.99%	98.11%	82.23%	60.80%
price, volume, tradeDate, tradeHour, tradeMinute, tradeSecond, bidEndUserRef	44	100.00%	100.00%	66.09%	65.11%
price, volume, tradeDate, tradeHour, tradeMinute, tradeSecond, bidEndUserRef, bidUser	63	100.00%	31.22%	84.25%	65.50%
tradeDate, tradeHour, tradeMinute, tradeSecond, bidEndUserRef, bidUser	61	100.00%	100.00%	92.26%	100.00%
price, volume, bidEndUserRef, bidUser	59	100.00%	99.84%	92.26%	75.79%

6.3 Clustering

Clustering is a kind of unsupervised learning and it is a good choice to use when the data is unlabeled. The purpose clustering is to divide all data points in the data into different clusters. The data inside of each cluster should contain similar data. Most clustering algorithms are trained with n data points with the purpose to find k clusters using some kind of metric for similarity between each of the data points. The ideal cluster contains data that are both compact and isolated. Because of the unlabeled data, one of the challenges with clustering is to set the correct value of k. Another challenge with clustering is that most algorithms that are commonly used are sensitive to noise[31].

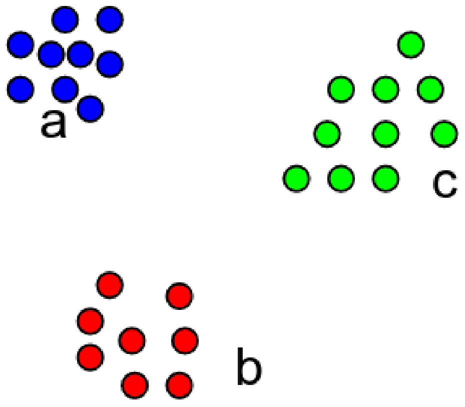


Figure 25: Example of three clusters

Three different clustering algorithms that are included in Spark MLlib are K-means, Bisecting K-means and Gaussian Mixture model.

6.3.1 K-means

The K-means algorithm is one of the most common clustering algorithms. K-means starts with randomly creating k cluster centers called centroids. Then each data point from the set is assigned to the centroid it is closest to. The centroids are then recalculated and changed so that they are in the center of all other data points in the cluster. Distance is calculated by squared Euclidean distance. The second and third step are repeated until it converges to the final clustering. The goal is to minimize the squared error distance for each data point to its cluster centroid[31].

Problems with K-means it that it often will have a runtime complexity that is at worst exponential. The other problem is that K-means might not always find the global optimum, it sometimes converges to a local optimum instead. The speed and how simple the algorithm is will make up for the problems though. A variant of K-means called K-means++ have been created to remove these problems by making optimal initialization of the initial centroids[3].

The implementation of K-means in spark uses a variant of K-means++ called K-means||. The K-means|| is parallelized and is used to select optimal initial centroids instead of random ones like in the original K-means. The initialization algorithm starts by randomly choosing one of the data points as a centroid. Then the other centroids are chosen by the rest of data points. It is calculated with probability proportional to the nearest already existing centroid and the data point. The k-means|| also uses an oversampling factor that is not used in the k-means++[3].

6.3.2 Bisecting K-means

An other algorithm in spark is the Bisecting K-means algorithm. It is a combination of K-means and Hierarchical clustering. It starts with one big cluster containing all data and then uses K-means with k set to 2 to split the cluster into two parts. This is then repeated for each new cluster that is created until it has produced k clusters[69].

There are two types of strategies that used in hierarchical clustering. This two types strategies:[59]

- Agglomerative or bottom-up approach, this approach will start in its own cluster, and pairs of with other clusters and merged as it moves to higher hierarchy.
- Divisive or top-down approach, this approach starts from one cluster, and it will split recursively as it moves down to lower hierarchy

Bisecting K-means algorithm in spark has used divisive or top-down approach.

Bisecting K-means is a lot slower than the original K-means because needs to run the K-Means algorithm several times before it converges. But it can often find better clusters and is sometimes more likely to find the global optimum.

6.3.3 Gaussian Mixture Model

Gaussian Mixture Model (GMM) is a composite distribution where points are drawn from one of the K gaussian sub-distributions where each of points will have own probability or a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussians distributions with unknown parameters [59][50]. Gaussian Mixture Model is an extension of K-Means algorithm. It is similar to Fuzzy K-means because GMM will produce probability result of each data point for each cluster. After all, Gaussian Mixture could be categorized as a density algorithm[71]. In this project, we use Spark MLlib as our machine learning library and this library has support by Spark. In MLlib, Gaussian Mixture Model will use expectation-maximation approach to induce the maximum-likelihood model[59].

Expectation-maximization is a method to find the maximum likelihood estimates in the presence of missing or hidden data[10][4]. This method divided into two different steps:

- E-step of expectation step is to find weights of each point and calculate the probability of membership in each cluster of each data point.
- M-step or maximization step is for each cluster to update the location, normalization, and shaped based on all data points, making use of the weights.

With the expectation-maximization approach, Gaussian Mixture model will not have a hard-edged sphere like in K-Means, but it will make a smooth Gaussian model [50]. However, this approach has the same problem like other other machine learning approach which is miss global optimal solution, nevertheless, we can overcome this problem by using random starting location for each running.

Gaussian mixture model has three different covariance type such as diagonal, spherical, full which can be seen in figure 26. These three types of covariance has their own degree of freedom in the shape of clusters. Diagonal covariance will determine a size of the cluster along each dimension can be set independently where resulting an ellipse constrained along with axes. Spherical covariance will make a shape of the cluster in all dimensions are equal, this method will have a result which similar to k-means, though it is not entirely equivalent. Full covariance is the last type of covariance in Gaussian mixture model which more complicated and computationally expensive. Full covariance will allow each of clusters to be modeled ellipse with arbitrary orientation [71].

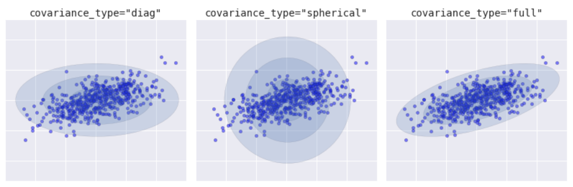


Figure 26: Covariance Type

In Gaussian mixture model, we will still observe overfitting problems which happens in other machine learning algorithms. We can solve this problem by doing cross-validation. Furthermore, there is another way we can do to avoid the overfitting by adjusting the model likelihood using Akaike information criterion (AIC) or Bayesian information criterion (BIC) [71]. This two methods can help us to determine a good number of cluster for our data. We can determine a good number of clusters by using elbow method from the result of AIC and BIC. In this project, we use MATLAB do to AIC and BIC analysis because Spark does not support this kind of analysis.

6.3.4 Anomaly detection

When the clusters are found using some clustering algorithm, the clusters can be used to find anomalies. The anomalies are discovered by looking at the outliers in the data. An outlier is a data point that lies far away from other data points and does not really belong in any cluster. There exist two different kinds of outliers. One is the kind of outliers that can be found inside of a cluster that has a point that has a bigger distance to the center of the cluster than most of all the other points in that cluster. The other kind of outlier are all points of a very small cluster that is also far away from all other clusters[76].

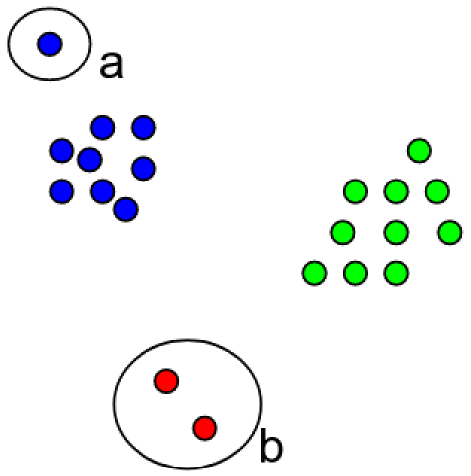


Figure 27: Outliers in Clusters

6.3.5 Implementation

In this project, we have implemented the K-means and Gaussian Mixture Model for anomaly detection. We also collect statistics from the clusters to be able to do analysis on the clusters.

There are several parameters in our clustering implementation that can be specified by the user. For these parameters, we are using java beans and the parameters can be found in ml-beans.xml and ml-models.xml. There are parameters that are used for all clustering algorithms and other that are only used for specific algorithms.

The parameters for all clustering algorithms are a number of clusters and number of features when using PCA. In K-means, the percentage of the data that would be considered as anomalies in each cluster needs to be specified and in GMM the threshold value for anomalies needs to be set.

The anomalies found using the K-means algorithm are calculated by using the squared Euclidean distance. The distanceThreshold parameter is used to decide how much of each cluster's members are considered to be anomalies. For the GMM we use the weight probability.

We use a java class UnsupervisedModel for the clustering. It can be used with all clustering algorithms that are implementing the UnsupervisedLearning interface. This means that new clustering algorithms like Bisecting K-means can be created by implementing the interface. There are three methods that must be implemented for all clustering algorithms. The buildModel method that is training the model, the getClusterDataset that returns the dataset of the trained model with the information of what cluster each data point belongs to and the showResult that will show the statistics of the clusters.

6.3.6 Experiments

We have done experiments with five different column from the Order Dataset. There are price, volume, hour part of timestamps, minute part of timestamps, and the second part of timestamps. We have used these attributes because Scila suggested us to use them. We have set some parameters for unsupervised algorithms ourselves when doing the experiments. We chose five as the number of clusters and three for the result of PCA. Weight probability in Gaussian mixture models has a value from 0 to 1. It means if weight probability for a data point getting closer to zero, it means that data point does not belong to that cluster[37]. We used 0.9 for distance threshold in K-means and 0.5 for weight probability threshold in GMM.

6.3.7 Result

In this experiment we useed the order data from 06 may 2017 and it consists of 347156 rows data for both KMeans and Gaussian mixture model algorithm.

The tables 7 and 8 show the results from the K-means experiments. For each cluster, we can compare the distance to the centroid for the outliers with the mean distance in that cluster. For example in cluster 1,4 and 5 we can see that the outliers have a much higher distance than the average of that cluster while in cluster 2 there is only a small distance. This means that the outliers in cluster 2 might actually not be outliers. We used the same threshold for all clusters and it might be better to use different because all clusters have different sizes and other important differences. Moreover, we can see from figure 29, there is some data which could be categorized as potential outliers from the SCILA data because it has spread far away from other data.

In the table 9 and 10, it shows the result of the Gaussian mixture model algorithm. It should return a weighted probability of each data points for each cluster. In this experiment, we have set weight probability threshold at 0.5. It means that if data points do not have weight probability with a value higher or equal than 0.5, it should consider the data point as ambiguous data of the cluster. As we could see at table 10, some of the weighted probability has distributed equally for two or more clusters. It means that data points would place in between two or more clusters. Moreover, It means that data points do not have uncommon properties for one cluster and we can categorize them as ambiguous data of the clusters. We give a flexibility to users for determined the threshold of weight probability because it would have different value while it needs to run with the actual data from SCILA AB. However, this weight probability could be useful while we have more information such as the approximate number of cluster, but, unfortunately, in this project, SCILA could not give that information to us. However, we could use another approach to attempt to find outliers in this data. We could determine the smallest cluster that could be an outlier cluster and in figure 30, the dark green cluster could be an outlier cluster because it was the smallest cluster than other clusters.

Table 7: Kmeans average distance from data points to centroid for each cluster

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
2.8637	7.0018	295.0512	5.7225	3.912

Table 8: Distance to centroid for outliers in each cluster

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
88.1736	12.2218	428.9979	79.812	92.296
75.1058	11.8306	356.2108	79.1606	90.6174
75.0593	11.3948	347.68	77.8769	90.3598
69.3013	11.125	347.6308	74.7163	89.3841
69.2294	10.7322	346.2992	74.2232	87.7603

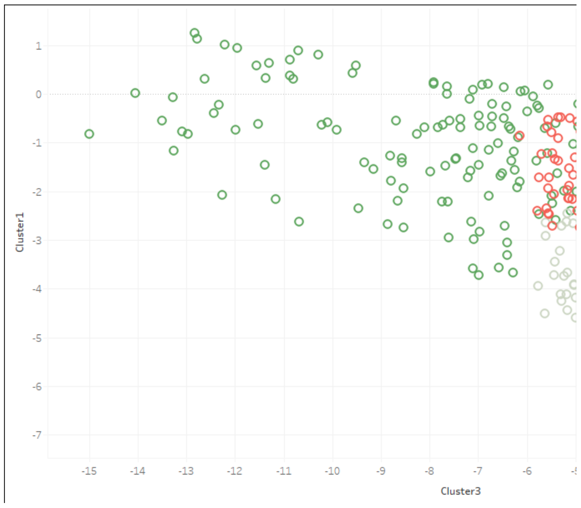


Figure 28: Result of K-Means

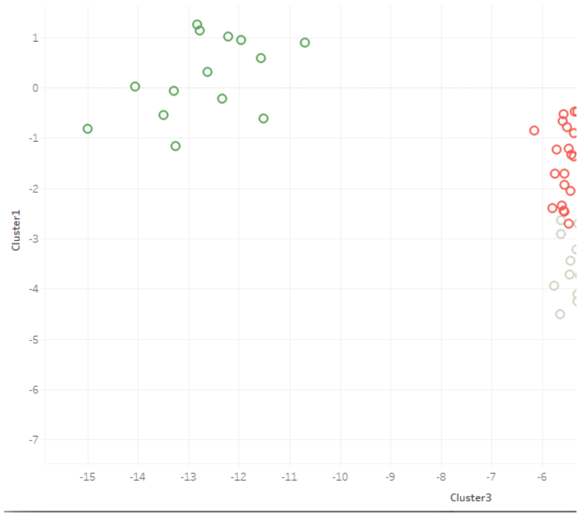


Figure 29: Result of K-Means Distance Calculation

Table 9: The Weight Probability of Gaussian Mixture Models

Data Point	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	0.0011	0.9968	0	0.0021	0
2	0.0012	0.9964	0	0.0024	0
3	0.0009	0.997	0	0.0021	0
4	0.001	0.9966	0	0.0024	0
5	0.0009	0.9971	0	0.0021	0

Table 10: The Weight Probability of Gaussian Mixture Models for Ambiguous Data

Data Point	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	0.4138	0	0.1283	0.4495	0.0084
2	0.4166	0	0.1293	0.4457	0.0084
3	0.0012	0.4061	0.2033	0.3887	0.0008
4	0.1744	0.4061	0.0006	0.4176	0.0012
5	0.0012	0.4061	0.2033	0.3887	0.0008

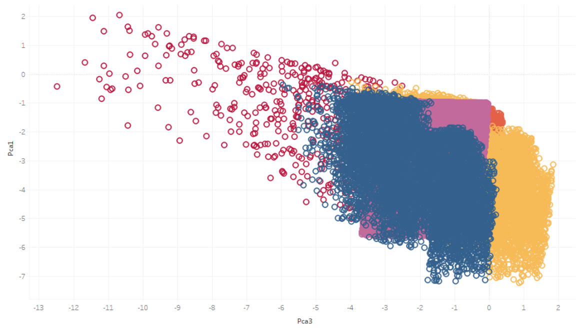


Figure 30: Result of Gaussian Mixture Model

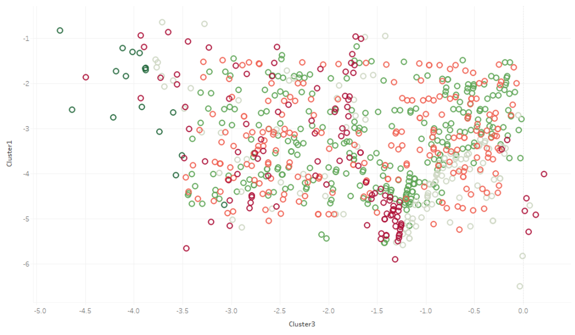


Figure 31: Result of Gaussian Mixture Model for Ambiguous Data

As we can see from figure 30 and 28, there is overlap between clusters from the data which was given by SCILA. It has made unsupervised learning become difficult to do. An other problem is that we had limited knowledge about the data from SCILA. We had problems to evaluate the anomalies that we found using the clustering algorithms. Since the data that we had was random and we did not know if the data contained anomalies or not. This way of doing clustering analysis would work better if we had one training set with no anomaly data inside and one test set that contained some anomalies. Then we would know that the model being trained correctly since training a model with data that contains noise can make a model produce wrong centroids. Moreover, we do not have any prior knowledge of how many clusters exists in this data or knowledge about any anomalies data which would be useful for confusion matrix. This confusion matrix would be useful to know the accuracy of our unsupervised algorithm.

6.4 Forecasting stock closing price

6.4.1 Introduction to Time series

Time series is a collection of data-points measured over a period of time.[45] Examples of time series can be opening/closing stock price, daily/monthly sales, yearly income, monthly revenue etc. Time series analysis uses various statistical methods to extract meaningful insights and characteristics from the time series data. [45] Time series analysis can be used to determine a trend from the data and also forecast by fitting appropriate models to it. [45] A typical time series would look like[55]:

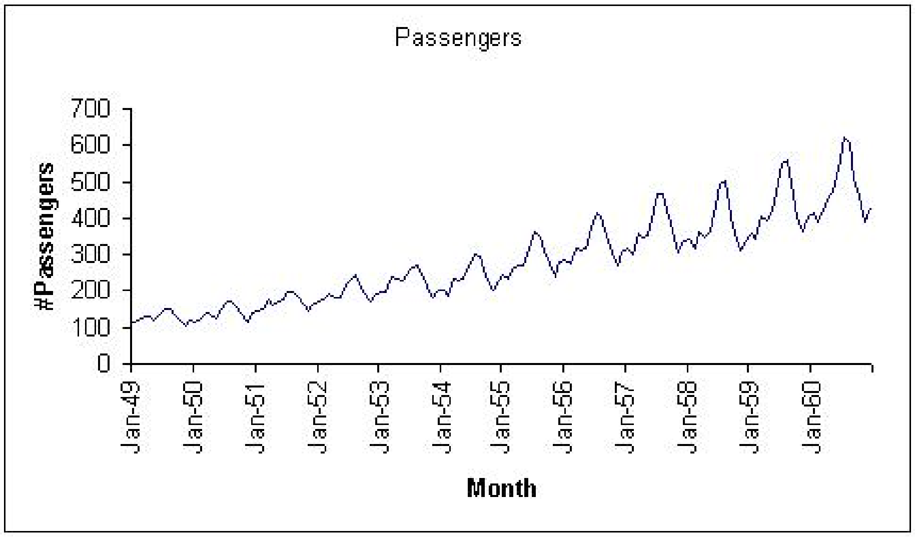


Figure 32: Passenger Data

6.4.2 Components of Time series

A time series can be decomposed into four elements:

1. **Trend**: When there is a long-term increase/decrease in the time series data then a trend exists.[25] The trend can be an increasing trend or a decreasing trend.

2. **Seasonality**: If the time series is influenced by seasonal factors such as the end of the month, a day of the week etc. then a seasonal pattern exists. [25] This seasonal pattern usually exists for a fixed period.[25]

3. **Cycles**: A cyclic pattern consists of a long-term irregular swings of the time series data. They usually last more than a year.[17] A cyclic pattern is usually observed in Business and Economical data.

4. **Residual**: The residual or the Error component of the time series does not contain any trend, seasonality or cycle.[25] It is a random fluctuation of data over a period of time. [17]

6.4.3 ARIMA Model

The term ARIMA stands for **A**uto-**R**egressive(AR) **I**ntegrated(I) **M**oving **A**verage(MA).[40] It is a forecasting technique which predicts the future value of time series data. It is considered as a short-term forecasting technique which requires at least **40 historical data points** in the time series.[40] It is sometimes called as Box-Jenkins method and works best when the time series data has a stable pattern without any outliers.[40] Arima Model is usually notated by ARIMA(p,d,q) where p,d, and q are the levels of AR, I and MA parts.[70] The main motive of the ARIMA model is to display a white noise or no pattern at all in the end which means the model should have extracted all the information from the data series and do forecasting.[70] Depending upon the time series data, the ARIMA model can be abbreviated as below:

1. When the model only contains the Auto-Regressive term, then it is abbreviated as AR model[49]
2. When the model only contains the Moving Average term, then it is abbreviated as MA model[49]
3. When the model does not contain the Integrated/Difference part, then it is abbreviated as ARMA model[49]

An ARIMA(p,d,q) Model can be represented as:[24]

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p} + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q} + e_t$$

6.4.4 Stationarity

Before moving into the details of ARIMA model, we need to know the concept of stationarity. A stationary time series is obtained when the mean and variance is constant over a period of time. Any time series with a trend or seasonality (non-stationary series) should be converted to a stationary series. This is because it is easy to predict a stationary series as its mean and variance will be same in the future also. Example of a stationary series is a white noise which does not have any underlying pattern.[6]

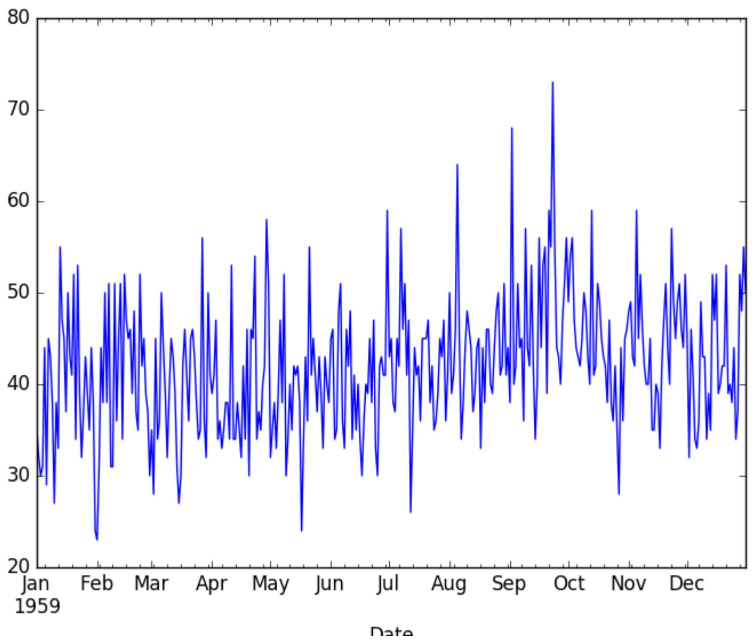


Figure 33: White noise

6.4.5 Integrated (I)

The first step in the ARIMA model is the Integrated part. In this part, if the time series data is non-stationary series with trend or seasonality then it is converted to a stationary series. Differencing is one of the common ways of converting a Non-stationary series to a Stationary series [70] It is done by subtracting the current time series with its lagged series.[40] Differencing can be performed as many times in the time series data till we get a stationary series. This is called as Order of differencing. A "first order difference" means the Differencing is performed only once. A "second order differencing" means that the result of the "first order difference" is further differenced again.[70] Depending upon the order of Differencing, the value of "d" in notation ARIMA(p,d,q) is set. The formula for different order of Differencing can be shown:[70]

1. *No Differencing(d=0)* :

$$Y_t^1 = Y_t$$

2. *1st order Differencing(d=1)* :

$$Y_t^1 = Y_t - Y_{t-1}$$

3. *2nd order Differencing(d=2)* :

$$Y_t^1 = Y_t - Y_{t-1} - (Y_{t-1} - Y_{t-2})$$

As an example, consider the below sales data after 1st differencing[70]:

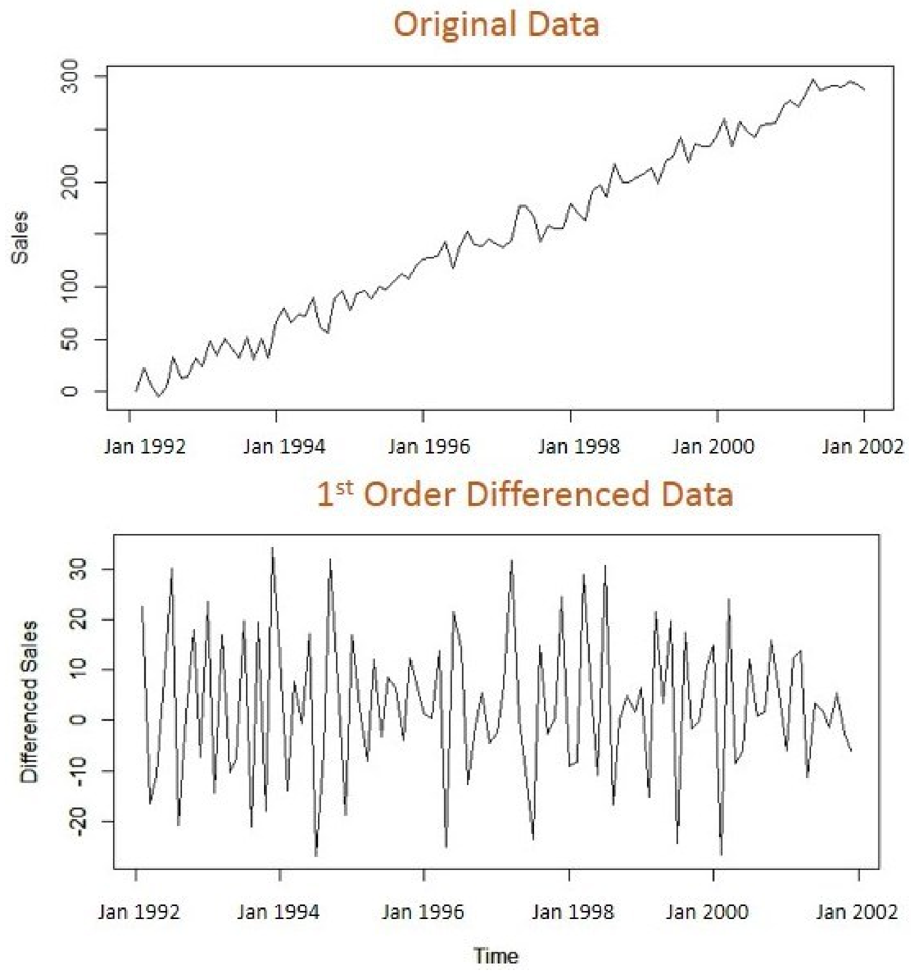


Figure 34: Time series after 1st order differencing

6.4.6 Auto-Regressive (AR) Model

Once the time series data is made stationary with the help of differencing, we need to check whether previous period's data values have any influence on the current period data values.[70] For example, April's stock market value can have an influence on the May's stock value. This can be done with the help of an Auto-Regressive Model. In this model, the forecast of the variable is be done using a linear combination of its past values.[22] An Auto-Regression Model of order p (AR(p)) can be written as:[22]

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p} + e_t$$

where c is a constant, e_t is the white noise and

$$\phi_1, \phi_2, ..., \phi_p$$

are the parameters of the model. The order p represents the number of lags in the model. For example, AR(1) or ARIMA(1,0,0) model can be represented as,

$$Y_t = c + \phi_1 Y_{t-1} + e_t$$

6.4.7 Moving Average (MA) Model

The Moving Average Model is also a Regression like model which extracts the influence of previous period's error term on the current period error term in a time series.[70] In this model, the forecast of a variable is done with a simple multiple linear regression of its past error values. A Moving Average of order q can be represented as:[23]

$$Y_t = c + \theta_1 e_{t-1} + \theta_2 e_{t-2} + ... + \theta_q e_{t-q} + e_t$$

where c is a constant, e_t is the white noise and

$$\theta_1, \theta_2, ..., \theta_q$$

are the parameters of the model. The order q represents the number of lags in the model. For example, MA(1) or ARIMA(0,0,1) model can be represented as,

$$Y_t = c + \theta_1 e_{t-1} + e_t$$

6.4.8 General Steps in ARIMA Model

1. **Time Series visualization**
   Visualize the time series to check for trend, seasonality or some random behavior.

2. **Stationarize the time series**
   If there is any trend or seasonality in the time series, we need to convert the series to a stationary series with the help of Differencing. As mentioned earlier, the original time series is converted to stationary series because it is easy to predict a stationary series as its mean and variance will be same in future also.

3. **Plot ACF/PACF graphs to find the optimal parameters for ARIMA Model**
   Once we get a stationary series, we need to identify two things:

   (a) Whether the time series is an AR or MA process?[68]

   (b) What is the order of AR or MA process?[68]

   The above questions can be answered with the help of an ACF(Auto-Correlation Function) plot. ACF plot is a total correlation between different lags.[68] For example, if the stock price at time t is x(t). Then the ACF plot shows the correlation of x(t) with x(t-1), x(t-2) and so on. [68]

   We know that Moving Average extracts the influence of previous period's error term on the current period error term. At lag n MA series, we will not get any correlation between x(t) and x(t-n-1).[68] So, the ACF plot will cut off at the $n - th$ lag which becomes easy for us to find the order of MA series. For an AR series, the ACF plot will gradually decline without any cut-offs. If the ACF plot decays slowly, then the series is not stationary.

   In order to find the lag for an AR series, we can use the PACF(Partial Auto-Correlation Function) plot. If the PACF plot cut offs at lag k, then it generally refers to AR(k) Model. If the PACF plot drops gradually, then suggests an MA Model.[21]

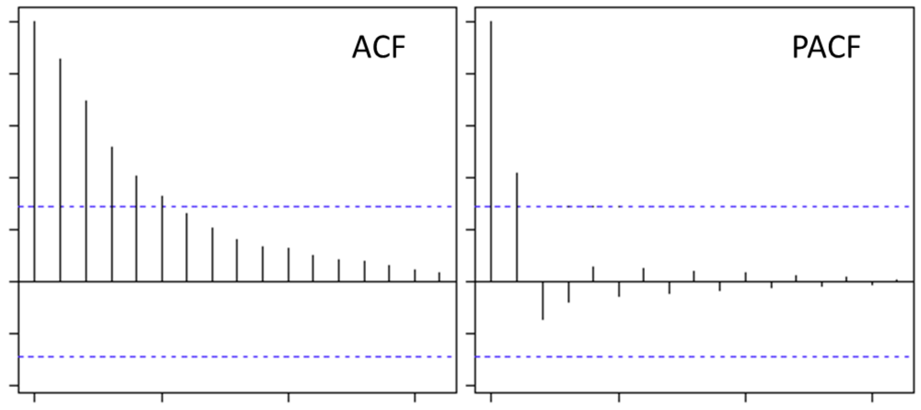


Figure 35: ACF and PACF plots

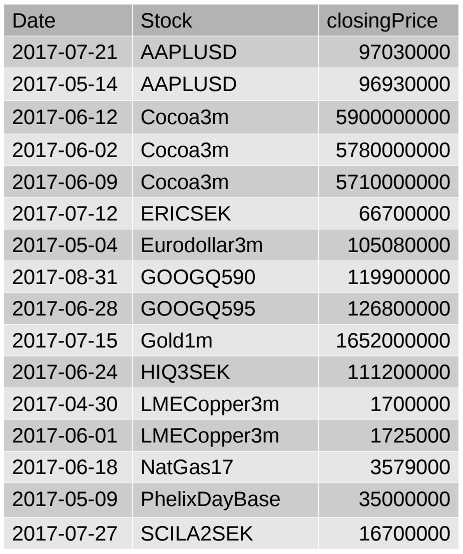
In figure 10, we can observe that the PACF plot cuts off at 2nd lag and the ACF plot drops gradually. This suggests that it is an AR(2) Model.

4. **Build the ARIMA Model**
   After we have found all the parameters for the ARIMA i.e p,d and q values, we can build the ARIMA model.

5. **Forecast the future values**
   Once we have built the ARIMA Model, we can forecast the future values of any time series. We can also visualize the time series with the forecasted values to check whether the model has given good prediction or not.[68]

6.4.9 Implementation

In this project, the ARIMA Model is implemented in R programming language. With the help of this model, we are going to predict the future stock closing price values. Stock closing price is chosen because it reflects all the activities of the index in a trading day.[2] The time series data which consists of a date, the stock's name and the closing stock price is generated from the Trade table of Scila data. Sample time series data is shown below:



Date	Stock	closingPrice
2017-07-21	AAPLUSD	97030000
2017-05-14	AAPLUSD	96930000
2017-06-12	Cocoa3m	5900000000
2017-06-02	Cocoa3m	5780000000
2017-06-09	Cocoa3m	5710000000
2017-07-12	ERICSEK	66700000
2017-05-04	Eurodollar3m	105080000
2017-08-31	GOOGQ590	119900000
2017-06-28	GOOGQ595	126800000
2017-07-15	Gold1m	1652000000
2017-06-24	HIQ3SEK	111200000
2017-04-30	LMECopper3m	1700000
2017-06-01	LMECopper3m	1725000
2017-06-18	NatGas17	3579000
2017-05-09	PhelixDayBase	35000000
2017-07-27	SCILA2SEK	16700000

Figure 36: Stock data

The above table is a snapshot of Scila stock data for May, June and July month. There are 39 different stocks in the Scila data. We are going to predict the future values for each stock using ARIMA model. This data is generated with the help of Apache Spark and ARIMA model is executed in R programming language.

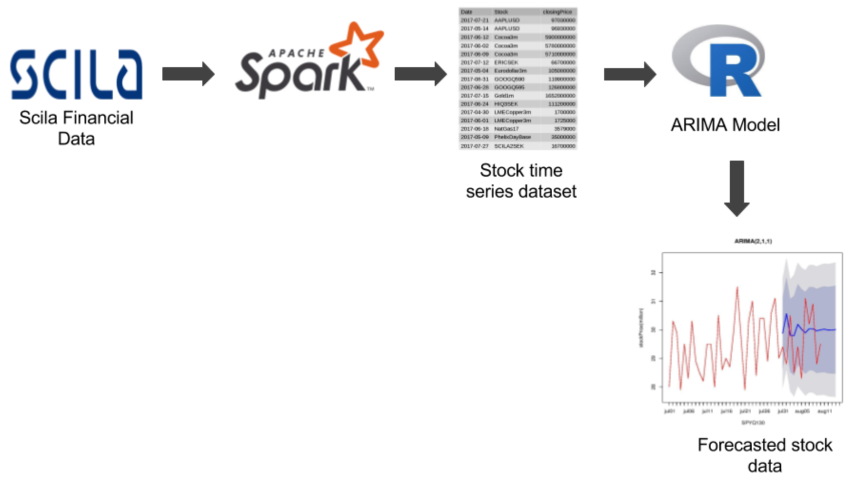


Figure 37: Flowchart for Arima forecast

The time series data which has been generated in Apache Spark is stored in an intermediate CSV file and this CSV file is used as an input for the R ARIMA script. The user has the freedom to choose the dataset to forecast such as a month or a year. Then the user also can change the number of days for forecasting. The data has been split based on the Stock names which will give us 39 independent time series datasets for each stock. For each independent stock dataset, the data has been divided into training(75%) and testing dataset(25%). It is recommended that the user choose a larger number of days to forecast with an increase in the size of the dataset. Since we split the data into train and test dataset, the model will not have enough days to forecast. So, the number of days to forecast should be greater than the test dataset in order to find the accuracy of the ARIMA model.

We can manually set the parameters i.e the orders of Auto-Regression, Integrated and Moving Average process for each stock dataset by looking at the ACF/PACF plots, but it is not feasible. So, we use *auto.arima*() function in R to fit the ARIMA model with the training dataset. This function finds the optimal parameters for different stock datasets. We can use the forecast() function to forecast the future stock values for each stock. We can then find the accuracy of the model with the test dataset. The accuracy() function in R is used to get the accuracy of the model which produces values of different error terms such as Mean Square Error(MSE), Root Mean Square Error(RMSE) etc. If we feel that the model has not given good accuracy, then we can manually set the parameters for ARIMA(p,d,q) model and improve the accuracy with the help of ACF/PACF plots. Currently, the R script produces two pdf files for visualization of forecasted data:

(a) Where the whole dataset is considered as one set and forecasted plots for every stock are produced

(b) Where the data is split into training and testing set. This can help us to get the accuracy of the model with the test dataset.

The figure 38 is a visualization of one of the stocks from Scila dataset with predicted future values:

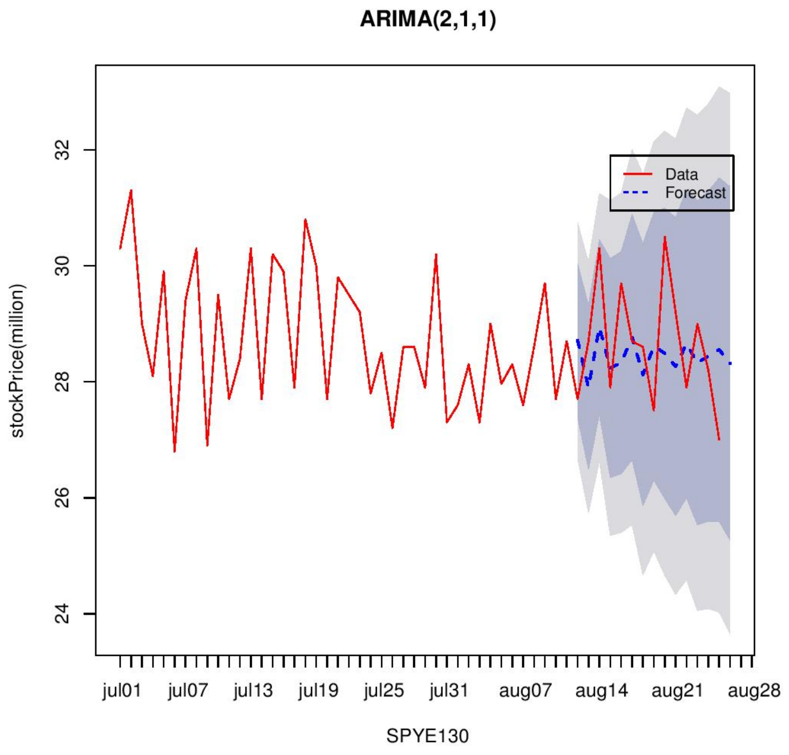
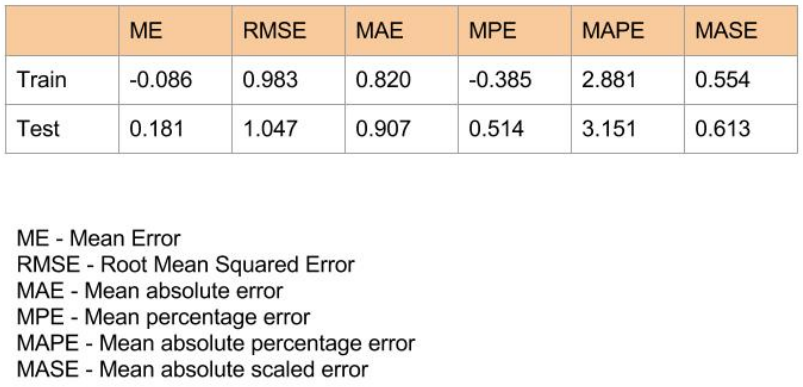


Figure 38: SPYE130 stock

The stock price (millions) is plotted against time for the stock "SPYE130". We have considered two months (July and August 2017) of data for the above visualization and forecasting is done for 15 days. Since it is not a large dataset, we are forecasting for few days. The red line in the above graph represents the original data and dotted blue line represents the predicted data by Arima model surrounded by confidence interval 80% and 95% regions. The above graph has an ARIMA(2,1,1) model where the parameters are automatically set by auto.arima() function. This means that the AR(Auto-regression) part has lag 2, the data series had to be differenced once to make the original series stationary (Integrated part is 1) and MV(Moving Average) part has lag 1. Below is the accuracy of the above ARIMA(2,1,1) model:



	ME	RMSE	MAE	MPE	MAPE	MASE
Train	-0.086	0.983	0.820	-0.385	2.881	0.554
Test	0.181	1.047	0.907	0.514	3.151	0.613

ME - Mean Error
RMSE - Root Mean Squared Error
MAE - Mean absolute error
MPE - Mean percentage error
MAPE - Mean absolute percentage error
MASE - Mean absolute scaled error

Figure 39: SPYE130 stock accuracy

The Root Mean Square Error of the stock "SPYE130" for test data is 1.047 which is good. For ARIMA model to give even better prediction, we need to have more data such as 4 or 5 years time duration. With more data, the model gives good prediction and accuracy.

6.5 Deep Learning 4 Java

6.5.1 What is DL4J?

To further expand on the Machine Learning part of the project we decided to investigate what different types of neural networks could possibly be used in association with the data provided by Scila. After we have done researching and testing, we determined that the neural network included in the Apache Spark standard library lacked a lot of functionality, only providing the use of a multi-layer perceptron neural network with limited options for configuration [58].

Looking beyond Apache Sparks provided machine learning library we decided to try implementing a neural network included in the Deep Learning 4 Java library(DL4J). DL4J is an open-source, JVM-based toolkit for building, training and deploying neural networks. It was built for Java and Scala and is well integrated with Spark. It provides tools to configure, train, and evaluate neural networks [35].

Compared to the Apache Spark neural network DL4J contains a wide range of different models and configurations of neural networks. Besides a multi-layer perceptron network used in Spark, it provides you the possibilities to set up a recurrent network, a long-short-term memory network and Restricted Boltzmann machines to name a few [34].

6.5.2 Data requirements for DL4J

All input to the deep learning networks – whether it's words, images or other data – must be transformed into numbers known as vectors, in a process called vectorization [32] [63]. The Deep Learning 4 Java networks require the training and test datasets to be in the DataSet data structure. DataSet differs from the Apache Sparks Dataset data structure and is part of the ND4J library. ND4J is a library associated with DL4J and is a library for using N-dimensional arrays in Java [43].

An ND4J DataSet consists of two INDArrays, which is the data structure for numerical arrays within the ND4J library. The first INDArray represents the features vector and the second INDArray represents the labels vector [41] [42]. The initialization of an INDArray is done with the ND4J objects create a method and requires the input to be a numerical and primitive Java data type [43].

6.5.3 Transforming our data from Spark to DL4J

Deep Learning 4 Java comes with an ETL [2] library called DataVec which purpose is to prepare the data to be used by DL4J. Unfortunately, none of the functionality in DataVec was suitable for usage with the data we already keep processed and cached inside Apache Spark, most data management in DataVec is designed to process data from disk storage [33].

Instead, we opted to design our own ETL process by taking advantage of some of Spark MLLib features, Spark User-Defined functions and Java 8 Utilities.

The below figure shows how the process from Spark Dataset to DL4J ready DataSet takes place. The gray objects represent the data structure of the data in that current stage. The blue objects represent the functions being called at that stage.

[2]Extract, Transform, Load: `https://en.wikipedia.org/wiki/Extract,_transform,_load`

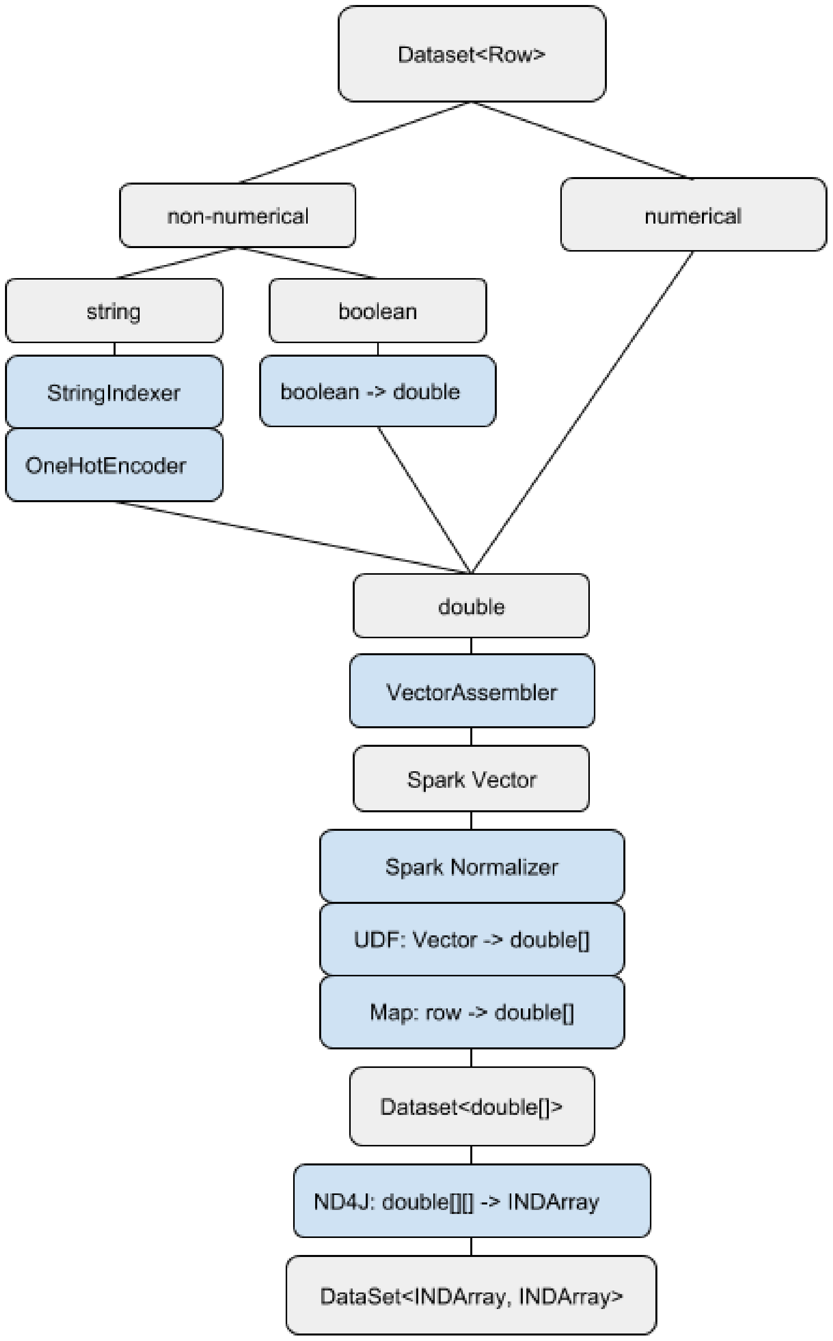


Figure 40: Flowchart depicting the general DL4J transformation.

6.5.4 DL4J Neural Network

While testing the DL4J data transformer and DL4Js neural network functionality we decided to implement a simple one layer feedforward neural network. The main goal of this of the neural network implemented as a proof of concept for DL4J and making sure that the data transformer worked successfully. Since the data provided was randomized and not optimal for machine learning we decided not to try and optimize or broaden the implementation.

7 Conclusions

When we were provided with the specifications, the main aim provided for us was to first see if we could utilize spark to parse the Scila propriety data format in manner that would be useful at a large scale, and secondly, use it then to create a spoofing detection mechanism and separately, use it in machine learning for anomaly detection. The broad range of the specifications allowed us to approach the problems on our own terms and we were able to arrive at satisfying conclusions which are mentioned below.

The main thing to understand about parsing the Scila data is that the JSON files cannot be read directly by Spark's API in order to create the datasets the SQL queries used in the spoofing detection expects. Due to the need to pre-process the input data, which slightly increases the time required in reading the data as the processing required increases.

There is a major difference in both memory usage and computation times when using cached data, parquet, the order in which and the method how the files are read and processed. The machine learning requires the data in a particular format while spoofing benefits from another, it is configurable to make a choice between two different implementations. One parses the data on a day-by-day basis while the other combines all the days into a much bigger dataset before performing any SQL queries.

By parsing the data on a day-by-day basis, some efficiency in Apache Spark's laziness is gained but performance in other areas is lost. Most notably is how the data can be cached more efficiently but more overhead to parse each dataset.

We were able to use SparkSQL to make filters for spoofing detection. It worked as per the requirements and in accordance with the specifications. After applying the filters we managed to obtain orders which fell amongst the criteria. The user can modify the parameters according to their discretion and ensure that they can widen or narrow their range. We found that there was no one specific parameter that could grantee and instead, required managing the parameters together.

We were able to utilize machine learning for this project in different approaches. The available dataset made it hard to actually distinguish some pattern, but we managed to do some experiments that could show a basic idea of what could be done using Spark and other available tools to be combined with it. Spark has a good library for machine learning, but it still has imperfection for supervised and unsupervised learning. However, there is nothing at all for time series algorithm. It does not have any time series model that could easily implement and that is why we need to implement time series model in R. In overall, spark MLlib is not as good as other libraries such as Scikit-learn and R. In clustering models, it has difficulty to do outliers detection. Moreover, it has not a give good result while we have used supervised learning.

8 Future work

There are still a lot of work that could be done in this project, but with the limitation of time and resources that we had made it impossible to do more. It would be great if the future works could use real big data and more directed goals. We suggest these future work for anyone who is interested in exploring more.

- Explore the latest technology such as Apache Flink for streaming and or batch processing.
- Implement the spoofing detection mechanism while utilizing spark streaming.
- Implement DBSCAN in Spark for Java to use as anomalies detection model.
- Implement Support Vector Machine that could handle the multiclass problem and use different kernels.
- Integrate Spark MLlib and DL4J to be able to explore more diverse types of neural network.

References

[1] Ben Aisen. *A Comparison of Multiclass SVM Methods*. http://courses.media.mit.edu/2006fall/mas622j/Projects/aisen-project/. (Accessed on 21/12/2017). Dec. 2006.

[2] Adebiyi A Ariyo, Adewumi O Adewumi, and Charles K Ayo. "Stock price prediction using the ARIMA model". In: *Computer Modelling and Simulation (UKSim), 2014 UKSim-AMSS 16th International Conference on*. IEEE. 2014, pp. 106–112.

[3] Bahman Bahmani et al. "Scalable K-Means++". In: (2012). URL: http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf.

[4] Sean Borman. *The Expectation Maximization Algorithm A short tutoria*. https://www.cs.utah.edu/~piyush/teaching/EM_algorithm.pdf. (Accessed on 17/12/2017).

[5] Dhruba Borthakur. *HDFS Architecture Guide*. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. (Accessed on 05/12/2017). Apr. 2013.

[6] Jason Browniee. *How to Check if Time Series Data is Stationary with Python*. https://machinelearningmastery.com/time-series-data-stationary-python/. (Accessed on 14/12/2017). Dec. 2016.

[7] Jason Brownlee. *Why One-Hot Encode Data in Machine Learning?* https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/. (Accessed on 19/12/2017). July 2017.

[8] Checkstyle. *checkstyle - Checkstyle 8.5*. http://checkstyle.sourceforge.net/. (Accessed on 08/12/2017). Nov. 2017.

[9] Checkstyle. *Google Java Style Guide*. http://checkstyle.sourceforge.net/reports/google-java-style-20170228.html. (Accessed on 08/12/2017). Feb. 2017.

[10] Yihua Chen and Maya R. Gupta. *EM Demystified: An Expectation-Maximization Tutorial*. https://www2.ee.washington.edu/techsite/papers/documents/UWEETR-2010-0002.pdf. (Accessed on 17/12/2017).

[11] CircleCI. *Overview - CircleCI*. https://circleci.com/docs/2.0/about-circleci/. (Accessed on 08/12/2017). 2017.

[12] databricks.org. *Reading JSON Files*. https://docs.databricks.com/spark/latest/data-sources/read-json.html. (Accessed on 21/12/2017).

[13] OpenStack Foundation. *OpenStack*. `https://www.openstack.org/`. (Accessed on 05/12/2017).

[14] OpenStack Foundation. *OpenStack Docs: Horizon: The OpenStack Dashboard Project*. `https://docs.openstack.org/horizon/latest/`. (Accessed on 05/12/2017). Dec. 2017.

[15] R Foundation. *R: What is R?* `https://www.r-project.org/about.html`. (Accessed on 05/12/2017).

[16] The Apache Software Foundation. *Apache Maven Checkstyle Plugin*. `https://maven.apache.org/plugins/maven-checkstyle-plugin/`. (Accessed on 08/12/2017). Oct. 2015.

[17] David Gerbing. *Time series Components*. `http://web.pdx.edu/~gerbing/515/Resources/ts.pdf`. (Accessed on 13/12/2017). Jan. 2016.

[18] Git. *Git - About Version Control*. `https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control`. (Accessed on 06/12/2017).

[19] Git. *Git - Git Basics*. `https://git-scm.com/book/en/v2/Getting-Started-Git-Basics`. (Accessed on 06/12/2017).

[20] hortonworks.com. *What is Tungsten for Apache Spark?* `https://community.hortonworks.com/articles/72502/what-is-tungsten-for-apache-spark.html`. (Accessed on 21/12/2017).

[21] Chih-Ling Hsu. *Time Series Analysis and Models*. `https://chih-ling-hsu.github.io/2017/03/20/time-series`. (Accessed on 19/12/2017). Mar. 2017.

[22] Rob J Hyndman and George Athanasopoulos. *Auto Regressive Model*. `https://www.otexts.org/fpp/8/3`. (Accessed on 18/12/2017). Sept. 2017.

[23] Rob J Hyndman and George Athanasopoulos. *Moving Average Model*. `https://www.otexts.org/fpp/8/4`. (Accessed on 18/12/2017). Sept. 2017.

[24] Rob J Hyndman and George Athanasopoulos. *Non-seasonal ARIMA*. `https://www.otexts.org/fpp/8/5`. (Accessed on 18/12/2017). Sept. 2017.

[25] Rob J Hyndman and George Athanasopoulos. *Time series Components*. `https://www.otexts.org/fpp/6/1`. (Accessed on 13/12/2017). Sept. 2017.

[26] Yoshiro Ikura and Mark Gimple. “Efficient scheduling algorithms for a single batch processing machine”. In: *Operations Research Letters* 5.2 (1986), pp. 61–65. ISSN: 0167-6377. DOI: https://doi.org/10.1016/0167-6377(86)90104-5. URL: http://www.sciencedirect.com/science/article/pii/0167637786901045.

[27] Docker Inc. *What is a Container*. https://www.docker.com/what-container. (Accessed on 06/12/2017). Sept. 2017.

[28] Docker Inc. *What is Docker?* https://www.docker.com/what-docker. (Accessed on 06/12/2017). Nov. 2017.

[29] intel.com. *Intel Hyper-Threading Technology*. https://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html. (Accessed on 12/01/2018).

[30] investopedia.com. *Financial Instrument*. https://www.investopedia.com/terms/f/financialinstrument.asp. (Accessed on 21/12/2017).

[31] Anil K. Jain. “Data clustering: 50 years beyond K-means”. In: *Pattern Recognition Letters* 31.8 (2010), pp. 651–666. DOI: https://doi.org/10.1016/j.patrec.2009.09.011. URL: http://www.sciencedirect.com/science/article/pii/S0167865509002323.

[32] Deep Learning 4 Java. *Custom Datasets*. https://deeplearning4j.org/customdatasets. (Accessed on 15/12/2017).

[33] Deep Learning 4 Java. *DataVec: A Vectorization and ETL Library*. https://deeplearning4j.org/datavec. (Accessed on 17/12/2017).

[34] Deep Learning 4 Java. *Documentation*. https://deeplearning4j.org/documentation. (Accessed on 15/12/2017).

[35] Deep Learning 4 Java. *Overview*. https://deeplearning4j.org/index.html. (Accessed on 15/12/2017).

[36] C. Ji et al. “Big Data Processing in Cloud Computing Environments”. In: *2012 12th International Symposium on Pervasive Systems, Algorithms and Networks*. Dec. 2012, pp. 17–23. DOI: 10.1109/I-SPAN.2012.9.

[37] Reid Johnson. *Cluster Similarity*. https://www3.nd.edu/~rjohns15/cse40647.sp14/www/content/lectures/14%20-%20EM%20&%20Evaluation.pdf. (Accessed on 08/01/2018).

[38] Holden Karau and Rachel Warren. *High Performance Spark. Best Practices For Scaling & Optimizing Apache Spark*. O’Reilly, 2017, pp. 7–26.

[39] The MathWorks. *MATLAB Product Description*. https://se.mathworks.com/help/matlab/learn_matlab/product-description.html. (Accessed on 08/12/2017). 2017.

[40] Jeff Morrison. *Arima Model*. http://www.forecastingsolutions.com/arima.html. (Accessed on 14/12/2017).

[41] ND4J. *ND4J Documentation: DataSet*. https://nd4j.org/doc/org/nd4j/linalg/dataset/DataSet.html. (Accessed on 17/12/2017).

[42] ND4J. *ND4J Documentation: INDArray*. https://nd4j.org/doc/org/nd4j/linalg/api/ndarray/INDArray.html. (Accessed on 17/12/2017).

[43] ND4J. *N-Dimensional Arrays for Java*. https://nd4j.org/introduction. (Accessed on 15/12/2017).

[44] Openstack.org. *Welcome to OpenStack Documentation*. https://docs.openstack.org/pike/. (Accessed on 21/12/2017).

[45] Research Optimus. *What is time series analysis?* https://www.researchoptimus.com/article/what-is-time-series-analysis.php. (Accessed on 13/12/2017). Oct. 2017.

[46] Johan Örtenblad. *MARKET SURVEILLANCE SYSTEM*. https://people.kth.se/~maguire/DEGREE-PROJECT-REPORTS/020606-Johan-Ortenblad.pdf. (Accessed on 1/12/2017). 2001.

[47] Sean Owen. *What are the differences between batch processing and stream processing systems? - Quora*. https://www.quora.com/What-are-the-differences-between-batch-processing-and-stream-processing-systems. (Accessed on 05/12/2017). Oct. 2014.

[48] Warren Sarle. *Section - How many hidden units should I use?* http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html. (Accessed on 21/12/2017). Mar. 2017.

[49] PennState Eberly College of Science. *Non-Seasonal ARIMA Models*. https://onlinecourses.science.psu.edu/stat510/node/64. (Accessed on 14/12/2017).

[50] Scikit-learn. *2.1. Gaussian mixture models*. http://scikit-learn.org/stable/modules/mixture.html. (Accessed on 15/12/2017).

[51] Scila. *Approaches to Market Surveillance in Emerging Markets*. https://www.iosco.org/library/pubdocs/pdf/IOSCOPD313.pdf. (Accessed on 1/12/2017). Dec. 2009.

[52] Scila. *Uppsala Student Project 2017 Financial Surveillance Using Big Data.* http://www.it.uu.se/edu/course/homepage/projektDV/ht17/specification.pdf. (Accessed on 1/12/2017). Aug. 2017.

[53] Pivotal Software. *Spring Framework Overview.* https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html. (Accessed on 05/12/2017). Nov. 2017.

[54] Tableau Software. *Tableau Desktop — Tableau Software.* https://www.tableau.com/products/desktop. (Accessed on 08/12/2017). 2017.

[55] Frontline Solvers. *Time series.* https://www.solver.com/time-series. (Accessed on 13/12/2017). Dec. 2017.

[56] Apache Spark. *Apache Spark FAQ.* https://spark.apache.org/faq. (Accessed on 19/12/2017).

[57] Apache Spark. *Classification and regression.* https://spark.apache.org/docs/latest/ml-classification-regression.html. (Accessed on 21/12/2017).

[58] Apache Spark. *Classification and regression.* https://spark.apache.org/docs/latest/. (Accessed on 15/12/2017).

[59] Apache Spark. *Clustering - RDD-based API.* https://spark.apache.org/docs/2.2.0/mllib-clustering.html#gaussian-mixture. (Accessed on 18/12/2017).

[60] Apache Spark. *Ensembles - RDD-based AP.* https://spark.apache.org/docs/latest/mllib-ensembles.html. (Accessed on 21/12/2017).

[61] Apache Spark. *Machine Learning Library (MLlib) Guide.* https://spark.apache.org/docs/latest/ml-guide.html. (Accessed on 08/01/2017).

[62] Apache Spark. *Overview - Spark 2.2.0 Documentation.* https://spark.apache.org/docs/2.2.0/. (Accessed on 05/12/2017).

[63] Apache Spark. *SQL Programming Guide.* https://spark.apache.org/docs/latest/sql-programming-guide.html. (Accessed on 15/12/2017).

[64] spark.apache.org. *Creating Datasets.* https://spark.apache.org/docs/latest/sql-programming-guide.html#creating-datasets. (Accessed on 21/12/2017).

[65] spark.apache.org. *Datasets and DataFrames.* https://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes. (Accessed on 21/12/2017).

[66] spark.apache.org. *Extracting, transforming and selecting features.* https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html#rdd-operations. (Accessed on 22/12/2017).

[67] spark.apache.org. *RDD Operations.* https://spark.apache.org/docs/latest/ml-features.html#vectorassembler. (Accessed on 21/12/2017).

[68] Tavish Srivastava. *A Complete Tutorial on Time Series Modeling in R.* https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/. (Accessed on 18/12/2017). Dec. 2015.

[69] Michael Steinbach, George Karypis, and Vipin Kumar. "A Comparison of Document Clustering Techniques". In: (2000). URL: https://pdfs.semanticscholar.org/c110/0f525044b2b926f7bd7f407ce7b0157bcfd8.pdf.

[70] Roopam Upadhyay. *Arima Model.* http://ucanalytics.com/blogs/arima-models-manufacturing-case-study-example-part-3/. (Accessed on 14/12/2017). June 2015.

[71] Jake VanderPlas. *Python Data Science Handbook.* Access from https://jakevdp.github.io/PythonDa
O'Reilly Media, Nov. 2016. ISBN: 978-1491912058.

[72] Liam Voughan. *How the Flash Crash Trader's $50 Million Fortune Vanished.* https://www.bloomberg.com/news/features/2017-02-10/how-the-flash-crash-trader-s-50-million-fortune-vanished. (Accessed on 20/12/2017). Feb. 2017.

[73] Wikipedia. *Multinomial logistic regression.* https://en.wikipedia.org/wiki/Multinomial_logistic_regression. (Accessed on 21/12/2017). Oct. 2017.

[74] Wikipedia. *Statistical classification.* https://en.wikipedia.org/wiki/Statistical_classification. (Accessed on 19/12/2017). Nov. 2017.

[75] Christo Wilson. *Guide to Using HDFS and Spark.* https://cbw.sh/spark.html. (Accessed on 21/12/2017).

[76] Kyung-A Yoon, Oh-Sung Kwon, and Doo-Hwan Bae. "An Approach to Outlier Detection of Software Measurement Data using the K-means Clustering Method". In: (2007). ISSN: 1938-6451. DOI: https://doi.org/10.1109/ESEM.2007.49. URL: http://ieeexplore.ieee.org/abstract/document/4343773/.

Appendices

Appendix A Installation Guide

A.1 Overview

All source code is available in the Github repository (it is a private repository as of now). The system has been tested on two different configurations a. cluster, b. docker image.

- Cluster:
  - OpenStack Infrastructure
  - Ubuntu trusty server 14.04.5
  - Oracle JDK 1.8.0_151
  - Apache Spark 2.2.0
  - HDFS 2.7.3 & 2.9.0
- Docker (without HDFS):
  - docker, 17.07.0-ce
  - Oracle JRE 1.8.0_151
  - openSUSE leap 42.3
  - Apache Spark 2.2.0

A.2 Setting up the cluster

In order to setup the cluster it is required to create a user to run the application, download and install HDFS and Spark and Java 8. We build one image and then clone it and spawn multiple VMs from that. The computer that will host Spark master and HDFS master will be called master and hdfs. The node that work only as workers in our setup are called cool-instance-{1, 2, 3, 4, 5, 6}.

If there is another setup steps A.2.3 A.2.3 A.2.4 A.2.5 A.2.3 A.2.4

A.2.1 User

We create a new user cluster in order to run both Spark and HDFS.

```
user=cluster
sudo useradd $user
sudo usermod $user -d /home/$user
sudo usermod $user -s /bin/bash
sudo mkdir /home/$USER
sudo chown $user:users /home/$user -R
```

Both Spark and HDFS need passwordless ssh access. The following commands will create an SSH key and will allow access to the same computer with the same key.

```
sudo su - $user
ssh-keygen -q -N "" -t rsa -b 4096
cp .ssh/id_rsa.pub .ssh/authorized_keys
```

A.2.2 Java

The application has been written in Java 8 and also both Spark and HDFS require Java to run. To download and install Java 8 on Ubuntu execute:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

A.2.3 Spark

Download Spark, add it to the bash path and configure it.

This command will download Spark version 2.2.0, will extract it under /usr/local/spark-2.2.0-bin-hadoop2.7, will create a soft link to spark and will give ownership to the previously created user.

```
sudo bash -c "curl -s https://d3kbcqa49mib13.cloudfront.net/spark-2.2.0-bi
        && cd /usr/local \
        && ln -s spark-2.2.0-bin-hadoop2.7 spark"
sudo chown $user:users /usr/local/spark -R
```

Create environment variable called SPARK_HOME to point at the base of Spark's installation and add Spark's binaries to the path. To take effect relogin from that user.

```
echo SPARK_HOME=/usr/local/spark >> ~/.bashrc
echo PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin >> ~/.bashrc
```

To set the output of the Spark to be Warning instead of Info or Error, to reduce log files.

```
sudo su - cluster
bash -c "cp /usr/local/spark/conf/log4j.properties.template /usr/local/spa
```

Set the host for the master and the port that the workers will listen on.

```
sudo su - cluster
echo "SPARK_MASTER_HOST='master'
SPARK_WORKER_PORT='15002'" > $SPARK\_HOME/conf/spark-env.sh
```

Set master's configuration.

```
sudo su - cluster
echo "spark.driver.port               15000
spark.blockManager.port         15001
spark.eventLog.enabled          true
spark.eventLog.dir              hdfs://hdfs:9000/shared/events
spark.history.fs.logDirectory   hdfs://hdfs:9000/shared/events
spark.history.ui.acls.enable    false
spark.history.ui.admin.acls     *
spark.history.ui.admin.acls.groups *" > $SPARK\_HOME/conf/spark-defaults.c
```

Set slaves on the *master* of the cluster.

```
sudo su - cluster
echo "localhost
cool-instance-1
cool-instance-2
cool-instance-3
cool-instance-4
cool-instance-5
cool-instance-6" > $SPARK\_HOME/conf/slaves
```

A.2.4 HDFS

For HDFS we both need to download, add it to the path and configure it. We download it from a mirror in Sweden, if that does not work for some reason you can find all available mirrors here.

```
curl http://apache.mirrors.spacedump.net/hadoop/common/hadoop-2.9.0/hadoop
         && cd /usr/local \
         && ln -s hadoop-2.9.0 hadoop"
sudo chown $user:users /usr/local/hadoop -R
```

To add it on the path execute:

```
sudo su - cluster
echo HADOOP_PREFIX=/usr/local/hadoop >> ~/.bashrc
echo HADOOP_COMMON_HOME=/usr/local/hadoop >> ~/.bashrc
echo HADOOP_HDFS_HOME=/usr/local/hadoop >> ~/.bashrc
echo HADOOP_MAPRED_HOME=/usr/local/hadoop >> ~/.bashrc
echo HADOOP_YARN_HOME=/usr/local/hadoop >> ~/.bashrc
echo HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop >> ~/.bashrc
```

To configure HDFS add these properties to $HADOOP_COMMON_HOME/etc/hadoop/hdfs-site.xml on all nodes.

```
sudo su - cluster
echo "<?xml version="1.0" encoding="UTF-8"?>
<?xml-styesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property
  <name>dfs.replication</name>
  <value>3</value>
</property>
<property>
  <name>dfs.permissions</name>
  <value>false</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>/usr/local/hadoop/hdfs/namenode</value>
</property>
```

```
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/usr/local/hadoop/hdfs/datanode</value>
</property>
</configuration>" > /usr/local/hadoop-2.9.0/etc/hadoop/hdfs-site.xml
```

Create necessary folders in all nodes:

```
sudo mkdir -p /app/hadoop/tmp
sudo chown $user:users /app/hadoop/tmp -R
sudo mkdir -p /usr/local/hadoop/hdfs/namenode
sudo chown $user:users /usr/local/hadoop/hdfs/namenode -R
sudo mkdir -p /usr/local/hadoop/hdfs/datanode
sudo chown $user:users /usr/local/hadoop/hdfs/datanode -R
```

Set hadoop settings on all nodes:

```
sudo su - cluster
echo "<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
  <name>fs.default.name</name>
  <value>hdfs://master:9000</value>
</property>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/app/hadoop/tmp</value>
</property>
</configuration>" > /usr/local/hadoop-2.9.0/etc/hadoop/core-site.xml
```

On all nodes add the hostnames in the Hadoop slave and master files:

```
sudo su - cluster
echo master > /usr/local/hadoop-2.9.0/etc/hadoop/masters
echo "master
cool-instance-1
cool-instance-2
cool-instance-3
cool-instance-4
```

```
cool-instance-5
cool-instance-6" > /usr/local/hadoop-2.9.0/etc/hadoop/slaves
```

A.2.5 System parameters

Add IP addresses with hostnames in /etc/host, substitute with your own IPs and hostnames.

```
sudo bash -c 'echo "127.0.0.1 localhost
10.10.33.83 master hdfs
10.10.33.89 cool-instance-1 inst-1
10.10.33.90 cool-instance-2 inst-2
10.10.33.91 cool-instance-3 inst-3
10.10.33.92 cool-instance-4 inst-4
10.10.33.93 cool-instance-5 inst-5
10.10.33.94 cool-instance-6 inst-6" >> /etc/hosts'
```

A.2.6 Prepare HDFS for Spark history server

Create The folders for the history server and add permissions.

```
sudo su - cluster
hdfs dfs -mkdir /shared
hdfs dfs -mkdir /shared/events
hdfs dfs -chmod -R 755 /shared/events
```

A.2.7 Upload data into HDFS

Create a folder for the data under root:

```
sudo su - cluster
hdfs dfs -mkdir /data/
hdfs dfs -chmod -R 755 /data/
cd %root_of_data%/
hdfs dfs -put 2017/ /data/
```

A.2.8 Starting the cluster

To start HDFS and Spark execute:

```
sudo su - cluster
start-dfs.sh
start-all.sh
start-history-server.sh
```

A.2.9 Executing the application

On a separate computer in the same network where only Spark is required (set up as shown above), copy myspark-submit to / and create a folder target (/target) containing all of the xml files and both Scila-$version$.jar and Scila-$version$-dependencies.jar. Then execute from the home folder myspark-submit. Now wait until the program finishes.

A.3 Docker

In this section we will demonstrate how to use the Docker image to use a simple cluster environment with a master and a worker. Adding more workers can be done either manually or by adapting the current script. These images are public available on Github, they operate with Spark and do not contain HDFS.

Tested on:

- openSUSE Tumbleweed 20171215
- Docker version 17.07.0-ce, build 87847530f717
- Ubuntu 16.04.3 LTS desktop
- Docker version 17.09.1-ce, build 19e2cf6

A.3.1 Docker image

This image is publicly available on Github (flanaras/docker-spark-configuration), which contains the Dockerfile for manual building. The base image for this contains Spark 2.2.0 and Oracle JRE8

Begin by cloning the repository by:

```
git clone https://github.com/flanaras/docker-spark-configuration.git
```

Build the images:

```
./build-docker
```

Create a folder called "docker-mount" on your home folder and place there:

- myspark-submit,
- a folder called target with all jar and xml files,
- a folder called data with the input data. The folder structure should like like %folder%/data/YEAR/MONTH/DAY

After all this has been done, it is possible to run the docker images to run the cluster. This can be done by invoking the ./run-docker script. The output will look something like this:

```
5434cc87dfaed1f526c318e0dcfe7978bda43d4206c0e4757fb0004d72e44947
7a0421e6fd8efb37f591f1fe6d45c6dac6c91daaa3585a68c0e7f99920f490c9
```

The first hash is the spark-master container and the second one is the spark-worker. Attach to the second one with: docker attach %hash%. Press any button to make the prompt appear. Now change directory to /host and then execute myspark-submit.

```
cd /host
./myspark-submit
```

To detach from the container without terminating it type ctrl p + ctrl q.

Some other useful docker commands are:

- docker ps -a (displays all the docker containers)
- docker system prune -a (delete all images, networks, etc from the computer)

Appendix B Troubleshooting

B.1 HDFS Troubleshooting

HDFS master has a web interface listening on port 50070 (http://host:50070/ by default and shows the current state of the cluster.

B.1.1 General

Do not upload data to HDFS while the application is running and vice versa.

There can be a lot of problems regarding HDFS how to make it run initially.

B.1.2 Take it online

To start the cluster type:

```
sudo su - cluster
start-dfs.sh
```

B.1.3 Take it offline

To turn off the cluster type:

```
sudo su - cluster
stop-dfs.sh
```

B.1.4 Report

To get an HDFS report from the terminal type:

```
sudo su - cluster
hdfs dfsadmin -report
```

B.1.5 Format

When the cluster starts it needs some time to verify its contents and show the cluster being online. Be sure to allow enough time for the cluster to perform this operation before formatting.

To format the HDFS cluster when it will not recognise all nodes or shows no space at all. Take the HDFS cluster offline. To format the namenode and datanode:

```
sudo su − cluster
hadoop namenode −format
hadoop datanode −format
```

After this take the cluster online and generate a report. If the report is still showing no space, take it offline again, delete all data from all nodes local data:

```
sudo bash −c ”rm −r /usr/local/hadoop/hdfs/datanode/∗”
```

After this format again and generate a report. This should solve the space problem.

B.1.6 Missing nodes

If there are some nodes missing from the web interface or the report there is probably a problem with the $HADOOP_HDFS_HOME/etc/hadoop/masters or $HADOOP_HDFS_HOME/etc/hadoop/slaves or the hostnames there are missing from /etc/hosts.

B.2 Spark Troubleshooting

Some common problems that might be occur follow.

B.2.1 Turn on cluster

To turn on the cluster type from the master:

```
sudo su − cluster
start−all.sh
```

This will put the master and worker nodes online and will serve a web interface on http://master:8080.

B.2.2 Turn off cluster

To turn off the cluster type from the master:

```
sudo su - cluster
stop-all.sh
```

B.2.3 Turn online specific node

To take online a specific node from the cluster, SSH to that computer and type:

```
sudo su - cluster
start-slave.sh spark://master:7077
```

B.2.4 Turn offline specific node

To take offline a specific node from the cluster, SSH to that computer and type:

```
sudo su - cluster
stop-slave.sh
```

B.2.5 Turn online history server

To put online the history server for statistics, SSH on the master and type:

```
sudo su - cluster
start-history-server.sh
```

This will start the spark history server on http://master:18080

B.2.6 Take history server offline

To turn off the history server

```
sudo su - cluster
stop-history-server.sh
```

B.2.7 Master will not start

If the master or worker will not start check the following.

- In the $SPARK_HOME/conf/spark-defaults.conf that both spark.eventLog.dir and spark.history.fs.logDirectory paths exist and are accessible.
- There is in /etc/hosts an entry for the hostname of that computer.
- That you have specified an IP address on one of the interfaces that are attached on the computer.

B.2.8 No entries in history server

On version 2.9.0 of Hadoop there was a problem with the permissions and there is no access to the to those files from the history server. To resolve this type from any node:

```
sudo su - cluster
hdfs dfs -chmod -R 755 /shared/events
```

B.2.9 Networking

In order for Spark it is required to have the following ports allowed in the firewall.

- 15000
- 15001
- 15002
- 7077 (master only)
- 8080 (master only, web ui)

- 18080 (master only, web ui)

Appendix C Usage instructions

C.1 General usage

The general usage of this prototype is done by changing the main bean that is stated in dev.xml as shown in the snippet.

```
<bean id="main" class="projectcs.control.mainrunnables.InitialScenario"/>
```

The only part that is needed to be changed is *InitialScenario*, and it could be changed into any of theses options:

- *InitialScenario* is for displaying initial scenario used in machine learning part, right now it is used as an example of making a scenario.
- *SpoofModule* is the main for spoof detection algorithm.
- *NeuralNetworkModule* is the main for DL4J implementation.
- *SupervisedScenario* is the main for classification implementation.
- *TimeSeriesScenario* is the main for ARIMA model implementation.
- *UnsupervisedLearningModule* is the main for clustering.
- *BenchmarkModule* is the main used for benchmarking the parsing process.
- *TableauReport* is the main for generating a CSV file for Tableau.

Other important part is a list of dates of data that will be used and it could be found in spark.xml.

```
<bean id="dateList" class="java.util.ArrayList">
        <constructor-arg>
            <util:list>
                <ref bean="wholeMonthExample"/>
            </util:list>
        </constructor-arg>
    </bean>
```

C.2 Spoofing

To make changes in the parameters for spoofing detection, they need to maneuver to the spoof.xml file and make changes in the following bean:

```
<bean id="spoof" class="projectcs.model.Spoof">
    <property name="minSpoofValue" value="4"/>
    <property name="spoofTime" value="500"/>
    <property name="participantLevel" value="enduserref"/>
    <property name="spoofCancelPerc" value="0.3"/>
    <property name="minPriceDifference" value="0.5"/>
</bean>
```

As of now the threshold for the minimum spoof value is set at 4 percent, while the spoof time threshold is 500 milliseconds, the level of the participant is limited to the enduserref string, the percentage limit for the spoof cancelation is 0.3 percent and for minimum price difference is 0.5 percent. We can change these values according to how we wide or narrow we would want to make the detection parameters.

C.3 ARIMA

In order to predict future stock price, we need to set the days to forecast in ml-beans.xml

```
<bean id="forecast" class="projectcs.model.ArimaForecast">
        <property name="forecastDays" value="10"/>
    </bean>
```

Currently, it is predicting 10 days but we can change the value according to increase in size of dataset. For example, we can set the forecastDays value = "20".
We also need to change the path of R script according to the user in ArimaForecast.java
:

```
public static final String FILENAME_R =
"/home/user/target/forecastStocks.r";
```

For example:

```
public static final String FILENAME_R =
"/home/rahul/target/forecastStocks.r";
```

C.4 Unsupervised Learning

In order to run unsupervised learning, it need to set some parameter in ml-beans.xml and ml-models.xml. In ml-beans.xml, consists of general parameters which will be use in both unsupervised learnings which are K-Means and Gaussian Mixture Models. The parameters need to be set in ml-beans.xml:

- Number of cluster This parameter has function to determine how many clusters that user wants. The defaults values for this parameter is 5.

  ```
  <property name="numberOfCluster" value= "5"/>
  ```

- Number of pca This parameter has function to determine how many dimensional will be created to run unsupervised learning. The defaults value for this parameter is 3

  ```
  <property name="pcaNum" value= "3"/>
  ```

- Clustering Model This parameter has function to determine what kind of unsupervised learning algorithm will be used.

  ```
  <property name="clusteringModel" ref="GaussianMixtureModels"/>
  ```

- Output Filename This parameter has function to determine the name of output file which has result of unsupervised learning algorithm

  ```
  <property name="outputFilename" value="clusterResultData.csv"/>
  ```

- Features This parameter has function to determine which columns will be used to run unsupervised algorithm. The name of column must be appear when user do initialize dataset for unsupervised algorithm. This parameter will receive array list of columns name.

  ```
  <property name="features">
      <bean class="java.util.ArrayList">
          <constructor-arg>
              <util:list>
                  <ref bean="price"/>
  ```

```
                        <ref bean="volume"/>
                        <ref bean="timestampHour"/>
                        <ref bean="timestampMinute"/>
                        <ref bean="timestampSecond"/>
                    </util:list>
                </constructor-arg>
            </bean>
        </property>
```

There is another parameters which can be modified to get different result of clusters. The parameter is placed at ml-models.xml. Each of unsupervised learning has its own parameters to be set. There are distance threshold which is used by K-Means and probability threshold used by Gaussian Mixture Models

```
<bean name="KMeansClustering" class="projectcs.ml.model.KmeansClu
    <property name="distanceThreshold" value="0.90000"/>
</bean>
<bean name="GaussianMixtureModels" class="projectcs.ml.model.Gauss
    <property name="probabilityThreshold" value="0.90000"/>
</bean>
```

C.5 Classification

There are 3 parameters used in general classification, which are classifier, label, and features. Available column to be used as label and or features is in ml-columns.xml and available classifier is in ml-models.xml. These could be changed in ml-beans.xml as shown in the snippet below.

```
<bean id="supervisedLearning" class="projectcs.ml.control.ClassificationPi
        <property name="classifier" ref="ml-svm"/>
        <property name="label" ref="allMember"/>
        <property name="features">
            <bean class="java.util.ArrayList">
                <constructor-arg>
                    <util:list>
                        <ref bean="price"/>
                        <ref bean="volume"/>
                        <ref bean="tradeMonth"/>
                        <ref bean="tradeDate"/>
                        <ref bean="tradeHour"/>
                        <ref bean="obId"/>
                    </util:list>
                </constructor-arg>
            </bean>
        </property>
    </bean>
```

The hyper-parameter tuning is a parameter that could be turn on or off in the ml-models.xml as shown below.

```
<bean id="tuneParameter" class="java.lang.Boolean">
        <constructor-arg value="false"/>
    </bean>
```

C.6 Generated Report for Tableau

Tableau cannot be integrated into Spark and vice versa, that is why we generated a CSV file that could be used in Tableau. The report is a predefined table based on the specification from Scila. It has parameter of starting date and ending date as shown in this snippet below which could be found in base.xml. Available dates could be found in dates.xml.

```
<bean id="makeReport" class="projectcs.model.GenerateReport">
        <property name="startDate" ref="10September2017"/>
        <property name="endDate" ref="20September2017"/>
    </bean>
```