Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Ville Vainio

Engineering analytics of big data pipelines for stock market data

Master's Thesis
Espoo, October 9, 2019

DRAFT! — December 5, 2019 — DRAFT!

Supervisor:	Professor Linh Truong
Advisor:	Professor Linh Truong

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Ville Vainio		
Title: Engineering analytics of big data pipelines for stock market data			
Date:	October 9, 2019	**Pages:**	67
Major:	Computer Science	**Code:**	SCI3042
Supervisor:	Professor Linh Truong		
Advisor:	Professor Linh Truong		

The world economics revolve around stock markets. The information in the stock market does not only represent the current state of economy but it also reflects other phenomenons that affect the prices of the stocks. This is why there are constantly new academic studies so that we could understand both global economy and these phenomenons better.

The size of the data that the stock market alone produces in a year is in the scale of terabytes. In order to analyze this data, tools that can handle this much infomation are necessary. Unfortunately, the information about these tools in the context of stock data is scattered, somewhat outdated and hard to find, which can drive away potential valuable research. The goal of this thesis is to provide information and tools that can make this analysis more accessible to data analysist.

This thesis performs a literary research on what is the current state of stock data analysis in big data environment focusing on historical data analysis pipelines and based on this research implements an open-source pipeline that anyone can use. The literary study shows that the current trend in stock data analysis is deep learning models giving the pipelines requirement to support these methods. In the implementation part we see that running deep learning libraries in distributed enviroment can be quite challenging and provide information about these challenges and how to solve them. The resulting pipeline does fullfil most of its requirements, but the results of the study are a bit incomplete without better comparative study.

Keywords:	stock market, big data, cloud computing, data analysis
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne- ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Ville Vainio		
Työn nimi: Osakemarkkina dataa käsittelevien big data järjestelmien tekninen analyysi			
Päiväys:	9. lokakuuta 2019	**Sivumäärä:**	67
Pääaine:	Tietotekniikka	**Koodi:**	SCI3042
Valvoja:	Professori Linh Truong		
Ohjaaja:	Professori Linh Truong		

Maailman talous pyörii osakemarkkinoiden ympärillä. Informaatio, jota osakemarkkina tarjoaa ei pelkästään kerro talouden nykytilasta vaan se myös heijastaa kaikkia osakemarkkinoihin vaikuttavia ilmiöitä. Tämän takia uusia tutkimuksia tehdään jatkuvasti jotta sekä taloutta että näitä ilmiöitä voitaisiin ymmärtää paremmin.

Osakemarkkinoiden tuottaman datan määrä vuodessa on teratavujen mittaluokassa. Jotta tätä dataa voitaisiin siis tutkia, tarvitaan työkaluja jotka pystyvät toimimaan tämän mittaluokan kanssa. Valitettavasti informaatio näistä työkaluista osake datan kontekstissa on hajanaista, jossain määrin vanhentunutta ja vaikeasti löydettävissä, mikä jossain tapauksissa estää mahdollisesti tärkeän analysiin tekemisen. Tämän diplomityön tarkoituksena on tarjota tietoa ja työkaluja, jotta tämän tyyppinen tutkimus olisi mahdollisimman helppopääsyistä tutkijoille, jotka eivät ole tämän alan ammattilaisia.

Tässä diplomityössä suoritamme kirjallisuuskatsauksen tämän hetken osake datan tutkimukseen big data ympäristössä. Keskitymme historialista dataa käsitteleviin järjestelmiin ja toteutamme avoimen lähdekoodin järjestelmän, jota kuka tahansa voi hyödyntää. Kirjallisuuskatsaus osoittaa että tämän hetken kehityssuunta osake datan tutkimisessa on deep learning menetelmät, jonka vuoksi järjestelmien on tuettava näitä metodeja. Työn kokeellisessa osuudessa huomaamme että deep learning ohjelmistokirjastojen käyttö hajautetussa ympäristössä voi olla hyvin hankalaa ja tarjoamme tietoa siitä mitä kaikkea voi mennä vikaan ja miten korjata nämä. Työn tuloksena saamme kehitettyä järjestelmän joka täyttää suurimman osan sille annetuista vaatimuksista, mutta tulokset jäävät hieman vajaavaisiksi, koska vertailevaa testausta ei ehditä toteuttaa.

Asiasanat:	osakkeet, big data, pilvilaskenta
Kieli:	Englanti

Espoo, October 9, 2019

Ville Vainio

Abbreviations and Acronyms

EMH	Efficient Market Hypothesis
RSI	Relative Strength Index
MACD	Moving Average Convergence Divergence
TF-IDF	Term Frequency–Inverse Document Frequency
HDFS	Hadoop Distributed File System
QoS	Quality of Service
DL4J	Deeplearning4j Machine learning library

Contents

Chapter 1

Introduction

The modern economy revolves around stock market. Stock market is a way for companies to obtain capital which they can invest into their own business. In exchange, the person who invests into the companies stocks technically owns a piece of the company which can return profit to the investor two different ways. The stock can grow in value, which allows the investor to sell the stock in higher price or the company itself can pay dividends to investors based on the number of stocks the investor owns from the company.

The price of the stock is simply determined by the law of supply and demand. If somebody is willing to pay a higher price for the stock then the price of the stock can grow. Because of this the stock market is in continuous fluctuation where people are selling and buying the stocks with the price they think the stock is worth using stockbrokers as the middleman. [37] All of this has lead to the question, how can we invest most optimally into stocks? This is where the following computational methods come in.

There are many strategies on how to invest into these stocks which depend on multiple factors such as; how much do you expect to profit with your investment, how much are you willing to take risk, do you want to make money by selling the stocks or by receiving dividends and so on. The underlying principle with every strategy is to minimize the risk you need to take in order to gain as much as profit as possible. Some of the strategies are based on subjective evaluation of the companies, but more technical strategies use metrics that are calculated from the financial statistics or the real-time market values. Strategies using the former data are called fundamental analysis and the strategies using latter data technical analysis. Neither of these approaches can predict the future of the market, but can statistically decrease the probability of larger losses in the market for the investor altough the probability of large losses is still not zero with these methods. [32]

Fundamental analysis is based on the idea that each stock has a intrinsic value that can be larger than the actual price of the stock in the market and buying these will eventually lead to profits.[22] The fundamental analysis focuses on the financial metrics that consist of companys overall statistics. These are for example how much the company has made profit, how much the company has paid dividends and what is companys cash flow. These tell a lot about the growth of the company and how the future of the company looks like. These metrics are usually published quarterly four times a year and present more long-term statistics about the company. Because of this, the amount of data these values present is quite small in terms of space.

The technical analysis that focuses on the real-time market values, on the other hand, needs new data almost daily. Stock exchanges are usually open from morning, opening around 8 to 10am, until evening, closing around 5 to 7pm on weekdays. Before and after this there are more limited pre- and after-hours trading which lasts usually around 1 to 2 hours depending on the exchange in which more limited stock trades can be made. During these hours multiple values are recorded on the prices of the stock from which the most important ones being: the highest price the stock was sold, the highest price the stock was sold and the number of stocks traded during the time interval. The technical analysis focuses on finding recognizable patterns through this data. [45] Where the data used by the fundamental analysis was relatively small, these values can generate gigabytes of raw data in a week.

Developing a system that can be used to conduct technical analysis means that the system should be planned to be able to handle large amounts of these data as time progresses. As such task is not trivial, the goal of this thesis is to provide developers and researchers, who want to analyse this data efficiently, basic knowledge and tools on what are the best current solutions on handling this data. With this knowledge these data scientists can save considerable amount of time without the need of trial and error when developing this kind of system from the ground up.

1.1 Application scenario

To have a better picture what are the challenges of developing a system for technical analysis are, we will next look at a example case. This is to give the reader more complete view of a system that handles stock market data analysis. After this, for the rest of the thesis, we are going to focus on a part of this system, but the purpose of this example is to give a better view of the system as a whole in order to understand some of the requirements that these other components give to the focus of this thesis.

Somewhat normal system for analysing both fundamental and technical

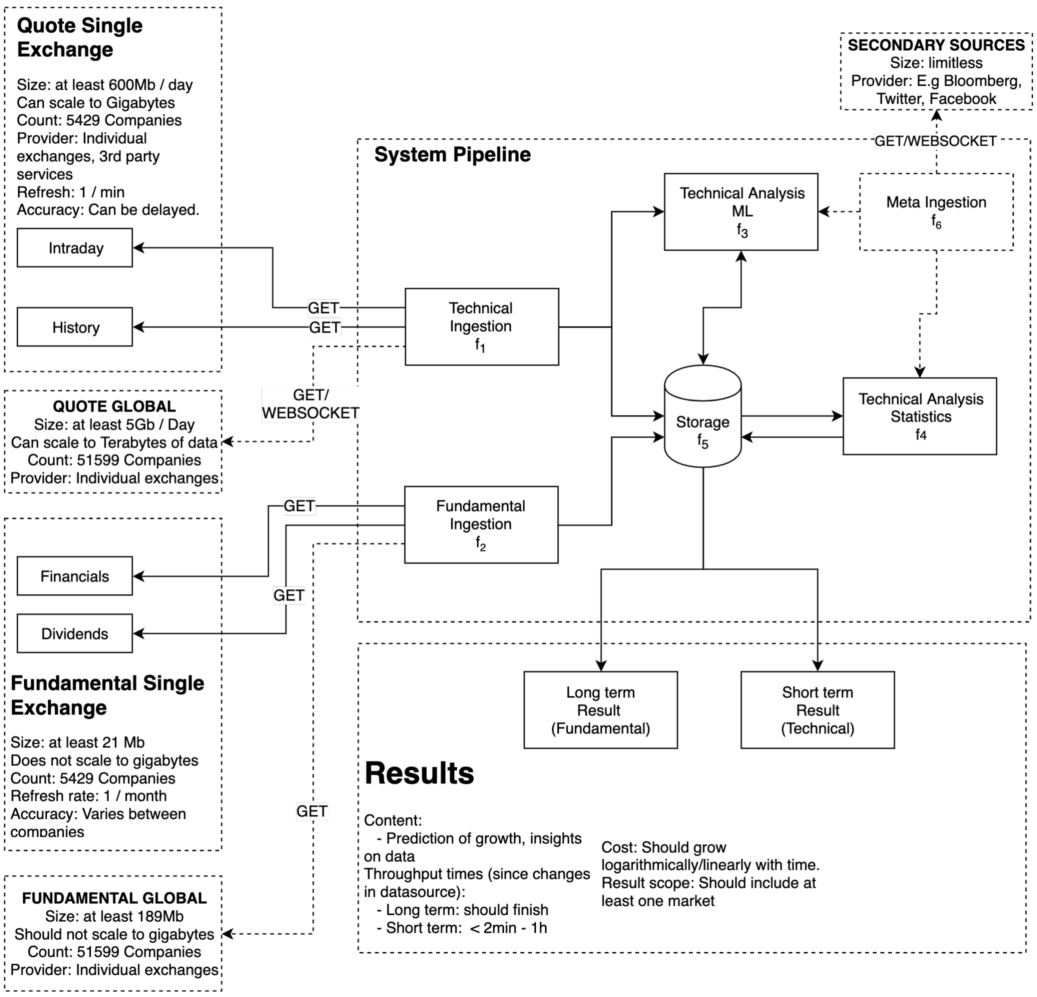
Quote Single Exchange
Size: at least 600Mb / day
Can scale to Gigabytes
Count: 5429 Companies
Provider: Individual exchanges, 3rd party services
Refresh: 1 / min
Accuracy: Can be delayed.
Intraday
History
QUOTE GLOBAL
Size: at least 5Gb / Day
Can scale to Terabytes of data
Count: 51599 Companies
Provider: Individual exchanges
Financials
Dividends
Fundamental Single Exchange
Size: at least 21 Mb
Does not scale to gigabytes
Count: 5429 Companies
Refresh rate: 1 / month
Accuracy: Varies between companies
FUNDAMENTAL GLOBAL
Size: at least 189Mb
Should not scale to gigabytes
Count: 51599 Companies
Provider: Individual exchanges
SECONDARY SOURCES
Size: limitless
Provider: E.g Bloomberg, Twitter, Facebook
GET/WEBSOCKET
System Pipeline
Technical Analysis ML f_3
Meta Ingestion f_6
GET
GET
Technical Ingestion f_1
GET/ WEBSOCKET
Storage f_5
Technical Analysis Statistics f_4
GET
Fundamental Ingestion f_2
GET
GET
Long term Result (Fundamental)
Short term Result (Technical)
Results
Content:
- Prediction of growth, insights on data
Throughput times (since changes in datasource):
- Long term: should finish
- Short term: < 2min - 1h
Cost: Should grow logarithmically/linearly with time.
Result scope: Should include at least one market

Figure 1.1: Example of a stock data pipeline

analysis is presented in figure 1. In the figure, the components marked with solid lines represent the core system functions, and the dashed lines represent add-on functionalities that should be possible to extend into the system in the future. The requirements of the system are usually as follows; The system should produce long (r_1) and short (r_2) term predictions of stock prices. The computation of long term predictions can take time because of the nature of these predictions but the short term predictions should be between two minutes to one hour available. Long term predictions are the result of fundamental analysis (f_2) and historical technical analysis (f_4) whereas the short term prediction should come mostly from quick technical analysis (f_3). The cost of the system should grow logarithmically/linear with time meaning that the cost of processing and storing data should not exponentially increase over time. Finally, the core system should be able to fulfill these requirements for at least the +5000 companies in the major U.S stock markets which is the most accessible and individually significant market.

The data to the application can be ingested from two main types of data sources quote and fundamental. These sources consist of values that were briefly described previously in technical and fundamental analysis respectively. The quote data is usually updated with minute intervals depending on the provider whereas the fundamental data does not change so often. Theoretically, the fundamental data can change anytime, because of dividends which can be payed whenever the companies want but this does not happen often and these values are mostly used in the long term fundamental analysis so longer update intervals are acceptable. In the figure, these are separated into the single market ones (d_1 and d_2), which represent the minimum of at least including the U.S stock market and the other sources (d_3 and d_4) that provide the same data on other global markets.

In the figure, the extendable global data sources are grouped into one box but in reality this data would be ingested from numberable different providers as there is no single entity at the time of writing this that provides all of this data. Theoretically the minimum of a maximum size of this extendable data would be 5Gb per day which is extrapolated in from the U.S market data based on statistics that in January 2019 there were globally 51 599 companies listed in the stock markets [61]. This amount can and will fluctuate as companies enter and exit the markets but it gives us the scale of data we are working with.

Today, stock market analysis has also a large focus on predicting stock prices using secondary data sources that can have reflect and affect the prices of stocks. These secondary data sources can be anything but at the moment one of the most researched sources are traditional media and social media data. Examples of using this kind of data to predict predict stocks can be found in [63], [55] and [44]. This is why the system should have the ability to extend to ingest data from arbitrary secondary sources (f_6 and d_5 in the figure) to provide more versatile predictions about the stocks. The amount of this data can be unlimited but is restricted to relevant sources.

Data is usually ingested from these data sources mainly using HTTP-protocol as this is the main method that these services (d_1 - d_4) provide. Other possible methods that are usually available are for example Excel sheets and sometimes websockets, of which the websockets can be actually useful in cloud system, but as the amount of data grows more custom file transfer methods are used. However, as the HTTP-methods are currently the most publicly available technology, this thesis is going to focus on these. The main ingestion functions are usually separated into two main types of functions f_1 and f_2. f_1 is constantly polling and processing data whereas f_2 handles batch processing. Both of these function usually store their raw output into the storage, but f_1 passes this also to the immediate technical analysis.

The system should usually have two types of methods on analysing data. For methdos that allow streaming updates there is f_3, which can for example be cumulative/reinforced ML models and for methods that need historical data in order to calculate the prediction, there is f_4. For fundamental analysis, there is no a specific function as the introduced data sources usually provide these values pre-calculated and these values are usually easy to calculate dynamically with little to none amount of processing.

Finally, at the center of the system, there is the storage which is used to store the calculated predictions as well as the raw data from the data sources for later analysis. For historical technical analysis, f_4, the storage should provide reasonable range query times when quering historical data and for the results the storage should provide efficient point queries for the results (< 1s).

1.2 Audience and Research Questions

We saw the possible complexity of a stock data analysis pipeline in the previous section. There are a lot of articles about how to build a stock analysis pipeline but these usually are focused on the analysis of small amount of data in order to test some hypothesis. Building a large-scale stock data analysis pipeline is a vastly complex task and to alleviate this process, this thesis plans to provide tools and information on how to build one.

This thesis is aimed at novice data scientists that work or want to work with stock market data, but do not have any experience in big data technologies. We define the term "novice" here to mean persons who have knowledge about analysing data using modern data analysis methods and know how to conduct this onto small datasets, but their proficiency lies strictly on the data analysis side. So the reader can be very good at analysing data, but lacks proficiency at the engineering side of big data technologies. The reason why the reader would want to read this could be because they have interest in the subject or are, for example, working for a startup which does not have resources to hire multiple persons with high degree on the subject. This thesis is meant for this kind of people, as the place to find easily knowledge on how to get started.

As the system can be very complex as seen in the previous section, we will be focusing on the part of the pipeline that calculates models based on historical data. This means the ingestion (f_1), storage (f_5) and analysis (f_4) of the historical data in a environment where the time restrictions are not that tight but the amount of data is enormous and the data can come from multiple different sources. We have chosen this part of the pipeline as this is the part that has the highest probability to scale to big data first and is the part of the pipeline that is usually referred when talked about data analysis on stock market data.

The main goal of this thesis is going to be to make the data analysis in this big data context more accessible within the time constraints and resources that this thesis has. What we mean by the word "accessible" here is that we want to minimize the time needed for the novice data scientist to start experimenting with their analysis, while making sure that developing models can continue smoothly in the future. In this sense, things that make the process of analysing stock data in big data context "inaccesible" are things such as scattered information about options, outdated information online and lack of information on how to integrate different parts together. This thesis plans to alleviate these aspects by providing aggregated information and an example case with solutions that can be used to navigate through these different problems.

The research questions that this thesis tries to answer are: What makes stock analysis possibly inaccesible?, What kind of challenges in practice novice data scientists face when building a modern big data pipeline for stock analysis? and How to build a pipeline that novice data scientist can use or benefit from?

With the first research question, we want to understand what are the problems with the current environment for big data analytics on stock data for novice data scientists. For this, we will be conducting literary study on the subject and we will analyse the results of this.

For the second and third research questions we will be conducting an experiment where we will build a pipeline using technologies that we think are realistic in real life and suitable for novice data scientist. We will be focusing on the way the pipeline is build and what are the challenges that we face during it that could gravely affect novice data scientists.

1.3 Structure of the Thesis

In the first chapter of the thesis we will be focusing on solving the first research question "What makes stock analysis possibly inaccesible?". We will define more precisely to whom this thesis is aimed at and examine what kind of challenges these people will face when starting big data analytics on stock market data.

We will then examine background information in order to understand better the underlying domain by by going through scientific papers about stock markets and stock analysis. We will examine trends and state-of-art methods on stock analysis that have been used in recent scientific research. Then we move on to examine what kind of pipelines exists in real-life. We will be examining both commercial and research pipelines in order to give a more complete picture on options that exist. We will also perform a literary research on technologies that are currently used to perform big-data ingestion, storing and analysis, selecting from the list of technologies mostly those that have been seen to be used in practice in this context while introducing couple of promising ones.

We will then start the experimental part of the research focusing on the two remaining research questions. In the first of these chapters we will define what will we be implementing, what are the goals of this experiment and how do we validate the results of this experiment. We will then start the actual implementation part. In the following chapter, we will describe what was done and what challenges we faced while implementing the systems. This chapter focuses mostly on the technical and practicalities that appeared during development.

After this in the final real chapter we move on to analyse the results that we got from the implementation chapter. We will try our best to validate the claims that we make about the example case and try to bring up fair criticism that the end result could possibly have. We then finish this chapter with a discussion on what could have been done better and how the systems developed could be improved in the future. Then finishing the thesis we will have a conclusion chapter that summarizes the results of the thesis.

Chapter 2

Motivation

We start this thesis by examining the problems that a novice data scientist faces when starting their process to implement big data stock market pipeline. In this chapter we will expand the problem introduced in the previous chapter and build the motivation on the experiments we will be conducting in the following chapters. We will start by defining the target group that these problems affect the most. These problems are not universal and to understand better to whom this thesis is benefitting the most we will be defining the target audience and the reasoning why would they be interested in the subject of this thesis. After this we will be defining the problem itself by dividing it to smaller parts and examining these parts individually.

2.1 Novice data scientist

As stated in the introduction, this thesis is meant for novice data scientists that have little to no experience on developing big data systems, but have general information on data analysis and want to use state of art methods to analyse stock market data. We use the term data scientist as it is usually used when talked about a person that does data analysis with enormous amount of data. This is in contrary to data analysist, which is in the same data analysis domain, but works with smaller amounts of data where one does not need to worry about optimization of calculations when the data is processed. [59] Because of this optimization aspect, data scientist must have knowledge not only about complex data analysis methods but also knowledge about developing and programming systems that scale.

Using these terms novice data scientist could be a data analysist which has a knowledge on algorithms and methods that are used to analyze reasonable amounts of data, but these methods do not scale with the data. As the tooling and methods are very different when the data scales, there are a plethora of things to learn and need for a lot of information. But how does then one with this kind of skill set find themselves in a position where they need to conduct data analysis on huge amounts of stock market data?

There is still a lot of hidden potential when it comes to stock market. With the arise of the amount of data from social medias and other new sources, there are new ways to analyse the stock data and at the same time there are new uses for the stock market data itself to understand these new sources of data. There is a lot of potential for new innovations when it comes to this data. Companies who want to benefit from these kind of innovations have their own highly professional teams that have resources to analyze this kind of data in large scale. However, the situation is different for smaller startups, which might have great ideas concerning the field but no resources to implement them. This is where the need for novice data scientists is and this is where one can find persons who benefit from this thesis the most.

As there exists a lot companies that have products to analyze stock data why would the reader then be building their own pipeline instead of using some ready-made product. Stock market analysis is a subject that has been researched for decades now and because of this there exists already a lot ready made tools for it which those who have enough money can use [16] [19]. However, with the rise of modern machine learning as the de facto way to conduct stock market analysis, these tools have not grown with this and are usually only made for statisticians or have very limiting capabilities concerning machine learning. This makes them not suitable for any data scientist to use who wants to try the latest methods and experiment ideas that have not been done before.

Other good question is that why would the reader then be interested in using methods to analyse the big data instead of using traditional methods that the data analysist would use. As the stock market data has been researched for decades, there is already a lot of knowledge about the stock analysis using the timeseries data with statistics. This is why the current interest is in using big data with machine learning to learn new things about this field which has been researched a lot.

2.2 Inaccessibility of big data stock analysis

Now that we have expanded our typical novice data scientist term and examined the reasons why would they be interested in the field of stock data in big data context, next we are going to examine what are the obstacles these types of people face when trying to start data analysis on stock market data using state of art methods. These are not problems that everyone who works in the field faces or do they mean that there is something majorly wrong with the current methods. These are more of a problems that can make it harder for new poeple in the field to get their ideas heard.

The problems can be seen to stem from three main factors; the information needed is scattered, the information needed is outdated and the needed information is lacking all together. We start by examining the obstacles in basic stock data analysis and then move to the technical side and examine obstacles in big data technologies to implement this analysis.

2.2.1 Obstacles in stock data analysis

As stated before stock data analysis is a subject that has been researched a lot and there exists a lot of materials on the subject that anybody can obtain. However, the material that is publicly available can be quite outdated compared to the state of art research. Much of the research is can be assumed to be carried by private companies which keep their findings as market secrets which is understandable taking into account their monetary value, but this makes it hard to not invent the wheel over and over again.

The papers published by researchers give us a better look at how, for example, machine learning is used in the stock market analysis. The problem with these papers is usually that they report their findings on a very high level usually focusing on theoretical side and keep their actual implementations private [41] [28] [36]. So the papers do provide valuable information on the subject but usually leave out important information how to reproduce the models which is vital information especially for the target audience of this thesis.

The problem is somewhat similar in the industrial side, although where in academic papers the theoretical side of algorithms is usually well explained, in industrial public information this side is usually left out. The information about industrial pipelines is very limited and usually the information available focuses on promoting some product that the pipeline uses instead of the pipeline itself [49] [56]. For novice data scientist this gives a climpse on how the industrial pipelines are implemented, but leaves a lot of details out on what are the algorithms these pipelines implement on data and how they are implemented in practice.

Availability of the actual stock data is also a somewhat of a problem. During the rise of internet, stock market data has been publicly available through services such as Yahoo Finance and Google Finance. However, as the price of the stock data has grown over the years both of these services have been shut down. [43] There exists some services which offer the same types of data, but there are still many papers quote the Yahoo Finance as their data source although it has been 2 years since it was shut down [54] [41]. So the needed stock data can be also hard to obtain.

2.2.2 Obstacles in big data frameworks

The situation of available information is much better on the big data tooling side. There are a lot of open-source solutions for many different use cases and these are usually very tested as they are. The problems arise when we have to filter the right technologies for the current domain as the amount of information is large. If one finally finds the right technologies to their use case, they are faced with a challenge to integrate and run these tools. In this section we are going to examine these problems.

For a novice data scientist the number of possible technologies you can use for stock data analysis can be overwhelming. The reason for this is that as there are a lot of technologies which all have different methods of solving their problems, none of these really stand out as the de facto solution for the stock market domain. So to choose the perfect solution for each use case can be a tremendous job as there are a lot of different factors to weight in.

This itself is not hard to overcome but what makes this really hard is that although the advantages and disadvantages of a particular technology for one specific purpose is well documented, the integration of one technology to a plethora of others is usually not. This is due to the constant development of each technology and the vast amount of different possible integrations. So it is really hard for a novice data scientist to pick technologies for their pipelines that seem to be the best solution individually while not leading to a deadend because the chose technologies do not work together.

Once the technologies, that seem to work with the problem instance you have, have been chosen the next step is to run them. This is usually documented very well for quickly starting the development on a single machine [3] [2], but these are usually instructions that are only meant for testing a single instance setup and are far from production ready ways to run the program or even develop it efficiently. To put this in other terms, the methods described are not sustainable in the long term. What this usually means is that the novice user either naively starts to develop their program on top these instructions that need a lot of revisions in order to run in production or have to spend a lot of time learning the framework and its nuances in order to build themselves a future proof development setup.

In this chapter, we examined the obstacles a lot of novice data scientists can face when starting their journey on big data stock market analysis. In the following chapters we will try to tear down these obstacles, while confirming that these problems do exist. Our goal is not to solve every problem that is listed here, but instead alleviate some of the burdens that these can introduce to a starting novice data scientist. We start by going through the necessary background information that is needed to build a big data pipeline that analyses stock market data.

Chapter 3

Background

In this chapter, we are going to look at the current stock market data analysis in big data environments. The goal of this chapter is to alleviate the problem with the scattered data of stock data analysis by aggregating information about the state of art stock market analysis. The other goal of this chapter is to introduce background information needed in the following chapters where we will be introducing the empirical part of this thesis to help reason the choices we make later. If the reader is familiar with these subjects we recommend moving to the next chapter as this chapter mostly aggregates existing information.

This chapter is divided into two parts. In the first part we will examine the current state of stock market data analysis in big data environment. This includes the methods and pipelines used to conduct this in real life. In the second part we will use the result of the this first section and study the most promising big data technologies that could be used to implement pipelines that analyze the stock data.

3.1 Stock market analysis

In order to build practical stock analysis systems, we start by looking at the methods of stock data analysis to understand the underlying domain. Stock market analysis is an enormous domain by itself and there are multiple ways to approach this data. For example possible problems to solve are finding anomalies in the data, predicting the future price and analysing the possible causalities. In this thesis we will be approaching this from the price prediction perspective which can also be used as a tool in other problems such as anomaly detection [36].

In this chapter we examine what are the state-of-the-art methods that researchers use to analyse stock market data. We are especially focusing on methods that in some way use big data for this analysis. We will be also inspecting real-life implementations of stock data analysis and examine some of the existing big data pipelines for stock market analysis focusing on what are the technologies used to build these.

3.1.1 Stock price prediction

The current methods on stock price prediction can be divided roughly into two different categories; statistical methods and machine learning methods. Both of these categories can be further divided into fundamental and technical analysis categories depending on what data they use as a source of analysis, but these categories overlap a little because of new ensembled models. With statistical methods we mean here the traditional mathematical models that need quite a lot of understanding of the domain in order to derive them. With machine learning methods, we mean algorithms and statistical models that derive underlying principles from data using patterns and inference.

Both of these methods are trying to beat the efficient market hypothesis (EMH). EMH states that the all the current public information available should already be seen in the price of the stock. In other words, the only thing that can affect the price of the stock would be unknown new information and the randomness of the system, which leads to a claim that stock prices can not be predicted using historical data. However, there have been multiple studies shown that this hypothesis could possibly be beaten using big data. [46] This hypothesis has also been challenged with overreaction hypothesis that states that the market overreacts to new information making it possible to somewhat predict the market before the prices change [29].

3.1.1.1 Statistical methods

The driving force of stock analysis in the past have been the statistical methods and there are still a lot of new papers analysing stocks using these methods. There are countless to approach this problem from statistician point of view so here we have listed only some of those that have some relevance to the subject of this thesis.

From the technical analysis perpective, where only the data related to the price of a stock is analysed, we will be looking at momentum indicators. [57] Momentum indicators, as the implies, measure the speed of change in the stock prices and investors make decisions based on the thresholds of these values whetever a stock is being overbought or oversold. [24] When searching research on big data usage in technical analysis there are some key indicators that show up frequently. These are relative strength index (RSI), moving average convergence divergence MACD and Williams %R which are all momentum indicators. These indicators can be used as they are but these have been also used as features that machine learning algorithms use to predics prices. [53]

All of these indicators measure the momentum of the price, but they all do it differently. RSI and Williams %R are both called oscillators because their values oscillate between maximum and minimum values they can get. Where as MACD compares long-term and short-term trends to predict if there is currently a notable trend. Because of the differences in the formulas and the factors that these values are measuring, the results from these formulas can be conflicting which can help to recognize one indicators bias when using multiple different indicators. [24]

Moving away from the momentum indicators, Autoregressive integrated moving average (ARIMA) models have also been successful way of analysing time series stock data and they have been also used with big data. [60] However, with the new advances in machine learning, there seems to better performance to be achieved with machine learning methods. [38] Due to this and the complexity of ARIMA models, we will only briefly make a note of them here and focus more on the machine learning methods in the next section.

In the fundamental analysis side, most of the recent developments have happened with textual data from news and social media using machine learning methods meaning that traditional statistical methods have not gained that much interest recently. However, as there is a lot of relevant financial big data that can be used with traditional methods, here are couple of recent examples of the usage of this kind data: Day et al. [29] tried to link oil prices to stock prices using global financial data streams with stochastic oscillator techniques. Kyo [40] combined technical and fundamental analysis by using regressive model that takes into account business cycles in Japan's stock market.

3.1.1.2 Machine learning methods

As stated before, machine learning is the current trending stock analysis methology that has gained a lot of interest. Both supervised and unsupervised learning techniques has been tried to predict the prices but recently neural networks especially have gained a lot of interest due to their adaptablity to any non-linear data. Neural networks have the advantage that they usually do not need any prior knowledge about the problem and this is the case when we are looking at seemingly random stock market data.

We start by looking at neural networks used to predict the market using only the market data (technical analysis data). There are multiple different neural network architectures to choose from and the most popular one currently seems to be a basic multilayered feed forward network (MFF) when analysing only the stock market data. There is some results that have shown that increasing the size of MFF, by adding layers and neurons, can produce better results. This however, increases the amount of needed computation and will probably have some upper bound before it starts to overfit the training data. [52]

Although MFF is seemingly the most popular, better results have been able to achieve with convolutional neural networks (CNN) and long short-term memory (LSTM). With convolutional neural networks, the upper hand to regular MFF seems to be the ability to express same amount of complexity with smaller amount neurons, although no direct comparisons have been made with these two. With LSTM however, it has been directly shown that it performs better than other memory-free machine learning methods including MFF. As stock data is literally a time series, the LSTM's ability to remember previous states seems to separate it with other methods. There currently seems to be no papers on how the LSTM performs compared to CNN so we can only assume that the performance of these two is pretty close when it comes to the complexity of the network. After these standalone architectures the next step to seem to be hybrid architectures combining more than one model into one and this has already achieved seemingly better results than these standalone models. [52]

In order to train a neural network to predict stock markets we need features and classes to label these features correctly. Because there is such a huge amount of data in stock transactions, manual labeling is not an option. The simplest way automatically generate features is to take calculate change in price in each time step. This way the network can for example output simple binary classification telling whetever the price is changing more or less than median price. [31] When we take this to a bit more complex level, similar features can be made from RSI, MACD and Williams %R values which we introduced in the previous section. [53]

When we move on to the fundamental analysis, similar kind of networks are used to with financial news and social media data. Again LSTM and ensemble models are used to connect this data from outside with the price trend of stock market. There are research using just general social media data that concerns the whole market but there are also usages of stock specific posts. Fundamental data that is in textual form, term frequency–inverse document frequency (TF-IDF) is a common choice to present features [42]. This data is then classified based on the general sentiment that they seem to represent. Positive sentiment toward company would lead to the rise of stock price and negative sentiment vice versa.

3.1.2 Existing pipelines

We then move on to examine the actual implemetations that do this analysis. For this section, we examined 19 different stock data analysis pipeline that handle or are able to handle big data. The big data might not be the actual stock data but for example social media data that was used to analyse the actual stock data. Information about these pipelines were all publicly available, although most of the pipelines were not open-sourced. Fourteen of these pipelines were reported in academic publications and the rest five are, or at least have been, in industrial use. There were also multiple open-source pipelines that have been developed for big data stock analysis, but information about the actual usage of these pipelines were not found. This is why we excluded these from this study in order to get more relevant results.

We have divided this section into subsections based on different parts from the pipeline starting from the furthest away of the possible clients and moving our way up towards the analysis phase. The biggest problem here is that the companies that work in the forefront of stock analysis, seldom share their software achitectures for business reasons. Nevertheless, with the public information available we can get an excellent view of the state-of-the-art pipelines in academic world and a general idea how the pipelines are build in the industry side.

It is also not that easy to compare technologies used in academia and industry. Both have different needs and goals that they wish to achieve. In academia, the pipeline is usually made in ad-hoc manner trying to minimize the time used for development and maximise the time needed for testing. This is possible, because researchers do not have same client side restrictions that industry might have. In industry side, it is usually important that the pipeline stays operational during the whole livecycle of the system. Where researchers only need these pipelines in order to conduct a couple of experiments, these industry pipelines have to survive possibly multiple of years of usage. These industry pipelines also serve the results to multiple clients that all expect small latencies from the system in order to use the system efficiently whereas in academia, this is not a requirement but helps the research to be made efficiently. So bearing these domain-specific requirements in mind, different technologies can be used to achieve optimal solution in different situations.

3.1.2.1 Data sources

Stock market data is not cheap. This is mostly because the exchanges that run the stock markets, usually make most of their profit with trade data and thus do not want to give this data freely away. There are however services who do provide part of this data with a much lower cost available to companies and researches which cannot possibly afford the complete real-time data. This partial data usually consist of historical end of day data, which is the aggregated statisctics of the stocks after the market has closed for the day. Although this data is complete in the sense that it tells all the necessary information about stocks price evolution on a day level, we will be referring this data as the aggregated data during this thesis and reserve the term complete stock market data for the data that contains all of the transactions. What this means for the systems is that the stock data can exponentially smaller at the beginning of pipelines lifecycle when there possibly is no way to access the complete stock market data, but system must be able to scale to this complete data set size when time progresses.

In academic papers, the Yahoo Finance API has been the de facto service used as the source of this partial stock data. [36] [28] [41] [53] Unfortunately, although this service have been used in papers published in 2019, this API was shutdown by Yahoo in 2017. Today, there are no service that provides the same amount of data that this API did, but some substituting services do exist.[43] These services will be introduced more completely in chapter 4 where we will evaluate which one to use in the implementation chapter.

3.1.2.2 Used technologies

We then move on to examine the actual technologies and frameworks that are in use by starting from the furthest away from the client, the ingestion layer. With ingestion we mean the part of the pipeline that handles fetching and initial processing of the data. This step with stock market data varies a lot depending who is fetching the data and for what purpose. The most common method for ingesting stock market data was found to be custom scripts. This is because the format of ingested data can vary greatly and scripts are relatively easy to write. This is usually enough with the aggregated stock market data, however, it does not scale.

Common big data technologies are usually introduced when the system has to ingest for example great amounts of textual data such as news and social media feeds. For these purposes, the most common technology in scientific papers is Apache Flume which is used, for example, in [50] and [27]. Another framework that is commonly reported in the ingestion step is the Apache Kafka streaming platform. [42] [26] Both of these are open-source frameworks which makes their usage relatively cost-free. In the industry side the usage of ready-made services from cloud providers are common. These services are for example the Google Dataflow as in [49].

After the ingestion, the data has to be stored. Stock market data is quite structured by itself but the data from third-party sources used to enrich stock data is usually not. This puts a restriction on what are the possible storage options available. Academic papers usually do not have to worry about this as the systems just have support ad hoc calculations but in the company context this becomes a crucial part of the system as it is the component that enables low latency responses without having to fetch data all the way from the original data source.

With big data systems there is usually two options which are either cloud platform specific products or open-source HDFS-based systems. This is also the case with stock analysis systems. From the cloud platform products, there are information on usage of Amazons S3 and Googles BigTable with BigQuery. [56] [49] In the open-source side HBase [34] and Cassandra seem to be the two most openly used database solutions.

Then as we finally have the data in control, we can apply some analysis to it. In the analysis phase, there is not that much variety in used technologies. Apache Spark with its MLlib library is dominating this field with its capabilities to process data efficiently. [36] [25] [42] [28] Where Apache Hive and Apache Pig were previously most used technologies, now Spark seems to be taking their place [56].

In the industry side, there is indications of Apache Storm being used with real-time classification [26]. The strong point of Spark is that same models can be used with Spark Streaming framework, but better latencies can be achieved with Storm making it possibly better choice for companies which have resources to implement two separate systems [39].

3.1.2.3 Monitoring

Finally we are going to examine one specific charasteristic which is usually ignored when pipelines are reported. Good monitoring is essential to ensure that the pipeline is running correctly and optimally. In the next few paragraphs we explain a bit more about the importance of monitoring as this is one of the areas that we will focus while trying to create a stock data pipeline.

None of the public sources on pipelines that were examined for this chapter reported exatly how they monitored their pipelines in practice. In cases where for example Spark is used, we can assume that the process is monitored using Sparks own monitoring tools which measure load and resource usages. These tools are valuable to measure the perfomance of the application, but they do not always tell the whole truth about the applications state.

With pipelines that analyze massive amounts of data, the quality of data is very important. This is not only when choosing what data source to use but the data must be kept from corrupting during the whole pipeline and to ensure this good monitoring is essential. Perfomance monitoring can pick up these corruptions if the corruption is large enough to make the program crash. However, in most cases this corruption can be only a change in values that would pass as an input. This would then lead to erronous results which could take quite a bit of resources to calculate.

So in order to make right decisions concerning models and save time while training, it is crucial to identify these kinds of problems early as possible and this is why we need good monitoring. In the next section, we will be looking more closely on the technologies that could be used to build a pipeline that can conduct modern stock analysis. We will also examine couple of different ways to monitor the system in order to prevent problems that were raised in this section.

3.2 Possible Big Data Technologies

In this section, we will examine more carefully the technologies that can be used to implement a stock data pipelines in the big data context and try to answer the second research question in the process. We have gathered here technologies that we saw used in existing big data stock pipelines in the previous section. We also added couple of promising frameworks that could be used in the pipelines but there is no public information about companies using them yet. To keep this section compact and more useful for majority of the novice data scientists that do not possibly have access to costly resources, we have included only the open-source solutions here leaving outside the services that cloud providers offer such as Amazon S3 for storage or Google Dataflow for ingestion.

3.2.1 Data ingestion

Here we will be using the same definition for ingestion as before; data fetching and initial processing which includes data validation. This is one of the crucial parts of the pipeline as it is responsible of turning arbitrary data into facts that the rest of the pipeline can use and rely on. This also makes it the hardest part to develop as the developer must understand the data well enough that these parts can work efficiently and prevent erroneous states in the later stages of the pipeline.

We have gathered here three main technologies that are usually mentioned when developers are talking about open-source data ingestion, but in reality all of these frameworks have a bit different tasks they try to fulfill. This makes comparing these technologies harder and it usually just means that the better technology here is usually the one that is better suited for the current problem, which does not make the other ones any worse than the selected one. On top of Apache Flume and Apache Kafka which were already used in existing pipelines, we have also included here Apache NiFi which has promising features that could be suitable for our use case.

3.2.1.1 Apache Flume

Apache Flume is one of the Apache Software Foundation (ASF) projects that is quite popular in the context of ingestion. From the projects official website we have definition that Flume is "distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data" [2]. So the main focus of Flume is to manage log data, but as we have already seen, this is not the only use case for this tool.

Flume was first introduced by Cloudera in 2011. In same year, it was officially moved under ASF and came out of the incubation phase the following year. During this incubation, developers had already started to refactor the Flume and the result of this is the 1.X lineage of Flume which is still going to this day. [35] At the time of writing this the latest production-ready version is 1.9.0.

Flume's high level architecture can be divided into 3 different parts: sources, channels and sinks which all run inside an agent which is an abstraction for one Flume process. Data is inputted to the system from the sources, then it goes through channels and is finally written into sinks. While going through channels the data can be processed using functions that Flume calls Interceptors. [35]

The main unit used of data in Flume is called Event which is structured like most of the other message formats you can find in network protocols. Event has a header, which is key-value pairs of meta data and a body which usually is the actual data. Overall, Flume architecture is message driven which allows it efficiently to multiplex data across multiple computing instances leviating the work load between these machines. [35]

When processing large amounts of data, it is important to avoid bottlenecks that can form in the pipeline and this is why knowing the amount of data flowing through Flume is extremely important. Bottlenecks that can form in Flume are for example cases where data is coming from sources faster than the Flume is able to write to sinks. These kind of situations can be avoided with the knowledge of the data domain when configuring Flume instances but also by monitoring Flume processes in order to respond to these kind of situations. [35]

3.2.1.2 Apache Kafka

Apache Kafka describes itself to be a ”distributed publish-subscribe messaging system” and it can be used for three different purposes: as a messaging system, as a storage system and as a stream processor. [3] Because we are examining Kafka in the context of data ingestion, we are mostly interested in its messaging and stream processing capabilities, but it’s good to keep in mind that it is possible to store data with Kafka for a longer time if necessary.

Before being a ASF project, it was first developed by LinkedIn to gather user activity data in 2010. [51] After being released from incubation in 2012 many other big industrial companies such as eBay and Uber [64] have taken kafka into use to manage their own enormous data systems. Today, Kafka is already in its version 2 and can be integrated with almost any modern big data framework. [51]

Kafka operates on publish-subscribe architecture where producers input data into the system by producing data and consumers subscribe to the data which they want to consume/receive. To make this work with big data, Kafka has a core abstraction called topic which has multiple immutable queues called partitions. When producers produce new data, this data is appended to a partition queue where where Kafka keeps track which items consumers have already consumed by tracking the offset of a consumer in a queue. With this kind of architecture Kafka makes promises that the data retains its order throughout the system and does not arrive to consumers out of order.

As for the scalability of this kind of system, a single partition must fit onto a single server, but topic which has multiple partitions does not have this limitation and this allows topics to scale over the Kafka cluster. Fault-tolerancy can be achieved by just replicating these partitions over the cluster. Multiple consumers can form consumer groups that consume some topic simultaneously from multiple partitions meaning that in addition of Kafka being itself scalable cluster, it allows its consumers to be also scalable cluster and without any additional complexity. [3]

3.2.1.3 Apache NiFi

Apache NiFi was made to dataflow management and its design is inspired by flow-based programming. It was originally developed by National Security Agency (NSA) as a system called ”Niagara Files” and was moved under ASF in 2014 making it the newest addition under ASF out of these three intoduced ingestion technologies. [23]

As we have seen already with the last two ingestion frameworks, all these frameworks have their own abstractions for data and the processes that handle this data. NiFi's core abstraction is FlowFile which represents the data that flows through the system. These are processed with FlowFile Processors that are connected to each other by Connections and these can be grouped into Process Groups. These processors and their connections are then managed by a Flow Controller which acts as the brain of each node in a NiFi cluster. [4]

NiFi cluster follows zero-master clustering paradigm meaning that the cluster does not have clear master nodes and vice versa no slave nodes. Every node in NiFi cluster processes data the same way and the data is divided and distributed to as many nodes as needed. Apache ZooKeeper is used to handle failure of nodes in the cluster.[4]

3.2.2 Data storage

Data storage forms the center of the pipeline and is one of the major places where bottlenecks can form. Possible bottlenecks are the query latency and the writing speed which can slow the whole application down if not implemented efficiently enough. If this non-trivial task was not enough, the data storage must also be able to resist error states that could occur when the hardware malfunctions for example with power outages or network outages. We have gathered three major big data storing technologies that were used in existing pipelines: HDFS, HBase and Cassandra. We start with the HDFS and move towards more industrially used systems.

3.2.2.1 Hadoop distributed file system

Hadoop distributed file system (HDFS), which is the core part of Apache Hadoop ecosystem, is one of the current defacto ways to store big data. It has gained a lot of popularity with the map-reduce programming paradigm. HDFS build so that it does not have to be run on high quality hardware, normal commodity hardware is more than enough to run the system.[13]

HDFS offers all the same functionalities that a traditional file system would offer. Clients can create, edit and delete files which can be stored into directories that form hierarchies. What makes HDFS differ from a normal file system is its ability to handle a lot larger data sets and at the same time have a better fault-tolerance than a traditional file system.[13]

HDFS is based on a master-slave architecture where the master is called NameNode and slaves are called DataNode. The NameNode stores and manages the state of the whole system and the actual client data never flows through this node. The data is stored into the DataNodes which manage this data based on the instructions that the NameNode gives them.[13]

For reliability, data is replicated between DataNodes so that the outage of individual DataNodes does not affect the overall perfomance of the system. In order to identify and react to these kind of failures, each DataNode send heartbeat-like messages to the NameNode, which then conducts needed actions if a heartbeat is missing. The NameNode itself, on the other hand, is a single point of failure. It does record the changes and the state of the system into files called EditLog and FsImage which can be used to replay the changes in the system if NameNode goes down temporarily and because of these files are crucial to get the system back up, usually multiple copies of these are stored into disk. [13]

3.2.2.2 Apache HBase

HDFS by itself is only made to store very large files, so the methods it provides our quite limited. This is where the Apache HBase comes in which is implemented based on Google's BigTable framework. Where BigTable uses Google's own file system, GFS, HBase is build on top of HDFS. HBase is a NoSQL database altough it does not natively have many features that a normal database would have. Notably, it does not have its own advanced query language.[33]

HBase's record structure is somewhat similar to its relational counterparts. Data is stored into rows that consist of columns, that are identified by a unique row key. Rows form tables and tables can be further grouped into namespaces. The difference to typical relational data model comes after this. Columns may have multiple versions and timestamp information about the column and its version are stored into separate entity called cell. The columns do not form table like structures with rows, and instead act like key-value pairs which can be grouped into column families. This way values that would be for example 'null' in normal relational database, do not take any room in HBase as such key-value pairs can be omitted. [33]

HBase's core abstraction of scalability is called Region. Region is a set of continous rows that are split when one Region grows too large. Each region is served by a single RegionServer and each RegionServer can serve multiple Regions. So Hbase cluster is scaled by adding these RegionServers which can serve more Regions to clients. [33]

HBase does not instantly store changes into disk. Changes are first recorded into a log called write-ahead log (WAL) and after this the change is stored into a memstore which is in memory. After the changes expire in the memory, the changes in memory are flushed into the disk as they are, into a file called HFile. In order to avoid large amounts of small HFiles in the disk, HBase periodically merges these files using a process called compaction. These are done in file scale (minor compaction), but also in larger scale where all the files inside one region are merged into one and data market for deletion can be cleaned in the process (major compaction). [33]

3.2.2.3 Apache Cassandra

Apache Cassandra is a NoSQL database that is build on peer to peer architecture. Cassandra provides its users same tools that a normal relational database would. Its record structure is the same with rows, columns and tables such as with typical relational database and it provides a SQL-like query language. What differs Cassandra from a normal relational database is its scalable architecture which we will be looking next. [62]

Cassandra's main scalable units are called Nodes that are a single instance of Cassandra running in a single machine. These Nodes are grouped together based on the data they serve and these form Rings. Data distributed and replicated between the nodes based on the hash of data's partition key. With consistent hashing algorithm, every Node has the same amount of data. [47]

Unlike HBase, Cassandra does not guarantee the consistency of data due to CAP theorem but instead guarantees the availability at all times. Cassandra implements "tunably consistent" paradigm where user can define for each write/read to be consistent with the cost of availability. This high availability, makes Cassandra better choise for example web application where the data must be available at all times, but the data doesn't necessary have to be up to date. [47]

3.2.3 Machine learning frameworks

When we researched existing pipelines we noticed that machine learning libraries such as Tensorflow and PyTorch are still quite used even when data can scale to enormous amounts. This is partly due to their support for processing data with GPU. This allows them to perform really well in single machine instances while needing minimal time to develop. But when these are run with data which size is scaled to terabytes, development becomes a bit more harder as this is not the environment these are designed to run at.

As we saw in the previous chapter, the Apache Spark framework with its Spark ML library is currently the most used big data machine learning platform. Spark, however, natively supports only classical machine learning models and currently has no native deep learning solutions, which are the state-of-the-art methods we are interested in. Spark is still a highly efficient processing framework for big data and the de facto technology in the market with its good integrations with the frameworks we already introduced and its mature ecosystem. That is why we are next going to briefly look at how the Spark ecosystem works and then move on to see tools which we can use to train deep learning models in Spark ecosystem without writing implementations from scratch.

3.2.3.1 Apache Spark

Apache Spark is a distributed in-memory data processing system that provides tools for scalable data processing. Spark has a master-slave architecture, where a driver node acts as a master and executes Spark program through Spark Context. The driver node passes tasks to worker nodes that the Spark Context together with Cluster Manager manages. The most popular cluster manager currently seems to be YARN but Mesos or Spark's own standalone could also be used for this job. Each worker node then has its own executor process which runs the given tasks in multiple threads when necessary.[5]

Until version 2.0, Spark's main programming interface was a data structure called Resilient Distributed Dataset (RDD). RDD is a collection of elements that can be partiotioned throughout the cluster allowing them to be processed in parallel accross multiple servers. RDD's are still in use for backward compatibility reasons, but from version 2.0 onward Spark's main abstraction has become structure called DataSet. DataSet is similar to RDD in high level, but it provides richer optimizations under the hood and higher level methods to transform the data. With the DataSets, a new abstraction called DataFrame was introduced which is can be compared to a table in relational database. DataFrame is a DataSet of Rows that can be manipulated with similar methods that you could do for DataSet.[5]

Spark has divided its functionalities into sub-modules that are specified into specific tasks such as Spark Streaming for stream processing and SQL for processing data in tabular form. We are mostly interested here in Spark ML module. Spark also has a module called Spark MLlib which some of the research still use. The main difference between Spark ML and MLlib is that

Spark ML provides newer DataFrame API whereas MLlib uses older RDD datastructures.[21]

Spark ML provides whole set of classical machine learning algorithms such as linear regression, naive bayesian and random forests. It also has a concept of pipelines which consist of different transformers and estimators, which are made to make model developing easier. However, the main problem with Spark ML is that it does not have natively tools to implement deep learning models on Spark.

3.2.3.2 Deeplearning4J

Deeplearning4J (DL4J) is distributed framework of deep learning algorithms that work on top Spark and Hadoop ecosystems. It provides all the most popular deep learning models such as multilayered networks, convolutional networks, recurrent networks. DL4J also provides a lot of tools for preprocessing the data before fed for training in the form of sub-projects. [10]

In distributed environment DL4J trains its models using Stochastic Gradient Descent (SGD). This is implemented in parallel on each node of the cluster using either of the two methods that the DL4J offers, gradient sharing or parameter averaging. From these two, the first one has become the preferred way to implement SGD in DL4J starting from its latest release and this is why we are going to ignore the technicalities of the latter one for now.

In the gradient sharing approach, the gradient is calculated asynchronously on each node in the cluster. The main idea behind DL4Js implementation of this, is that not every update on the gradient is sent to every other node. Each node has a threshold which defines when a change is large enough to be shared with the global gradient. This combined with its own heartbeat mechanism provides efficient and fault tolerant way of training a model in distributed environment. [10]

In academia, DL4J is not that used as its main language is Java, but this choice of language makes it really suitable for industrial use. However, DL4J supports Keras model import which allows user to use other languages than the JVM based alternatives. This makes python based prototype systems be able to run in industrial Spark cluster. [10]

3.2.3.3 Apache SystemML

Apache SystemML is a bit different machine learning system when compared to Spark ML and DL4J. Similarly to DL4J, SystemML also runs top of Spark, but unlike DL4J it is not a straight forward programming library. SystemML has its own R- and Python-like declarative machine learning languages (DML), which can be used to define machine learning algorithms that run on the Spark. [6]

Currently, these languages cover about the same use-cases that Spark ML does. What makes SystemML differ from Spark ML is that deep learning models developed in Keras or Caffe can be converted into DML. So theoretically SystemML supports deep learning through these libaries although its core methods do not have these methods. This allows it to be run on classical command-line interface, but also from jupyter notebooks which are currently vastly in use in academia. [6]

3.2.4 Monitoring

We have already seen quite a bit of framework specific monitoring solutions, which cover monitoring individual components in the pipeline. Next we will take this monitoring a step further and look at monitoring solutions that work outside of these components and can be used to monitor the global state of the pipeline. This can make monitoring easier when all the information is available in one place and it also helps to infer cause-effect relations.

We have gathered here two different monitoring solutions for two different needs. The ELK stack for monitoring the data flow inside the system and individual components and the ModelDB to specifically monitor the Machine learning models that are produced by the pipeline.

3.2.4.1 Log monitoring

The most common solution for monitoring logs in bigger system is the ELK stack. ELK stack which is an acronym for Elasticsearch, Logstash and Kibana stack, is a common stack used to implement collection of logs and their visualization in Big Data environment. The need for such as elaborate stack for just collecting logs, comes from the fact that when you have a big data system, the logs of such a system form a big data problem of their own, so in order to solve this problem this stack was developed. It provides scalable data ingestion system, with a distributed storage that can be accessed and visualized in efficient manner. [11]

In this stack, Logtash, an open-source data ingestion pipeline tool, is used to ingest the log data. This ingested data is the stored into Elasticsearch which is a distributed search and analytics engine, which provides REST interface. Finally, the data stored into Elasticsearch can be visualized and monitored with Kibana, which is a tool developed to do just this. [11]

Because of different user needs the stack has evolved into stack that the company behind these technologies calls Elastic Stack. This stack really only differs from ELK-stack by a component called Beats, which can be used to build more lightweight stack for simpler needs. However, pure ELK stack is still very popular option to handle log data. [11]

3.2.4.2 Machine learning monitoring

Open-source monitoring tools that are specifically designed for machine learning are not that common, but couple of tools do exist. These tools do not only help monitor the models perfomance, but also provide tools for deploying and versioning these models just like developer would use git to manage their code base.

ModelDB is a system developed in Massachusetts Institute of Technology (MIT) for ML model management. It provides tools that can be used to monitor the performance of models and compare these results with each other. It also allows logical versioning of models and helps to reproduce the models this way. ModelDB, however, does not support models from many different ML libraries and currently the only supported libraries are Spark ML and scikit-learn. The library is said to have second version coming up, but at the time of writing this the library has not had major update for an year. [58]

Another, newer option is an open-source project called MLFlow. It does all the same things that the ModelDB does, but markets itself as a more end-to-end solution. On top of tools for the managing explained in ModelDB paragraph, MLFlow puts more emphasis on packaging the models and the deployment of these models into actual use. [17]

Unlike ModelDB, MLFlow does not care about what is the library that generates models. It does this by providing CLI and REST API's that can be integrated with the system ignoring the underlying technologies. For convenience, it also provides APIs for Python, R and Java which can be used for tighter integration. MLFlow is as project quite young having its first full version released in June 2019, but it has a healthy development community and is actively developed. [17]

That concludes the needed background information needed to build a pipeline for historical stock market analysis using big data. In the next chapter, we will

Chapter 4

Planning

Next we will look at how do we can use the technologies in the last chapter in order to build a pipeline that can be used to examine stock data. We will be reviewing, what we will be doing in the implementation part of this thesis and analyze the information that we have seen to this point. Finally, we will look how we will evaluate the end-product and how is it better or worse than other alternatives.

As stated in the chapter about obstacles, there are multiple points that can hinder the development of big data pipeline for stock data analysis for novice data scientist. In this chapter we will start to prove that the current default methology is not suitable for novices and try to find a way which would suit novice data scientist better. We will be building an example pipeline using methods that would suit novices better than the default methods and show how hard this can be.

We will start by providing information about the options that the novice data scientist have for stock data source in 2019 as this is one of the real requirements for getting started as you need to have some data to analyse. Then we introduce our way of building a big data pipeline for stock analysis using technologies introduced in the previous chapter.

4.1 Data source

Most of the data sources concerning stock data are closed and heavily priced. However, there exists services that offer data for smaller scale development and analysis. A list of this kind of services is presented in the table 4.1. In the table the cheapest possible plan for each service is presented to give a better understanding what kind of possibilities there are for acquiring stock data for analysis purposes and what kind of data formats these offer.

Table 4.1: Stock data sources

	Type of data	Price	Data Structure	Restrictions
IEX [14]	Intraday and Historical (15 years)	0\$/month	JSON	500k data-points/month
Alpha Vantage [1]	Intraday and Historical (20+ years)	0\$/month	JSON/CSV	5 requests/min and 500/day
World Trading Data [20]	Intraday and Historical (from 1980)	0\$/month	JSON/CSV	5 symbols/request, 250 request/day and 25 intraday requests/day
Intrino [15]	Intraday	52\$/month	JSON / CSV / Excel	120 request/min
Quandl [18]	Historical EOD^1 (from 1996)	$15\$/month^2$	JSON / Excel	none
Barchart [8]	Intraday and Historical (6 months)	0\$/month	JSON / CSV / XML	25 symbols /request and 400 requests/day

1 EOD = End of Day

2 For academic use

From this table we can make couple of notes. For aquiring free data test data fast, the IEX provides the best option as its limits are not restricted to time intervals, but the 500k datapoint restriction does not provide nearly enough data for any serious application. For academic use, Quandl provides at the moment of writing this, the most affortable API for historical data analysis. Historical data is usually cheaper because with historical data alone you usually cannot make money as stock market revolves around the most recent data, but for training models and other data analysis it is ideal. Other thing to note is that the amount of historical data can vary greatly between services from 6 months (Barchart) to 39 years (World Trading Data).

For this thesis we have chosen to use data from IEX API which was an open API until 30.09.2019 when the company decided to close this in order to capitalize with the closed API which free plan is in the table. This data is from the 5 year interval between 2014 and 2019 and the relevant values that it contains are opening prices, closing prices, highest prices, lowest prices and volumes. In order to test freely with this data, we will implement a simple server that serves this data into our pipeline.

4.2 Goal of the implementation part

As we saw in the first chapter there is a lot of higher level information available about stock pipelines and the technologies they use, but as these are only high level information a lot of practicalities needed to reproduce the pipelines are left out. There is also a lot of good documentation about how to get started with these technologies but these methods usually are not that sustainable. What we want to do in the following implementation chapter is to bring this gap down by building a novice friendly pipeline that somewhat reflects what are the pipelines used in reality.

Other important thing we want to highlight is the challenges faced when integrating the possible technologies. We want to show what are the challenges that novices face and provide information from this in order to prevent these adversities on happening to our possible reader.

The pipeline will not be the best one could build for stock analysis, but the idea is more to show a way to build a novice friendly pipeline and highlight challenges that novice can face while building one. We will use a lot of ready made products listed in the previous chapter as we do not want to invent the same thing over again when somebody else has already done it better. So when we say we are building a pipeline, it is more that we are building a configuration to integrate all these pieces together.

4.3 Building a novice friendly pipeline

In the implementation chapter, we will build a pipeline that has two major characteristics to make them sustainable and more novice friendly; The pipeline will be entirely containerized using Docker and the whole pipeline can be operated using Scala. In this section we will go through the reasoning of these choices and how we think they solve some of the problems presented in the first chapter.

One of the problems with stock data research was that the pipelines presented were quite hard to reproduce as they were presented from such a high level. This combined with quickstart tutorials which encourage to run the technologies manually from command line can make it really hard for novice users to develop pipelines that are reproducible and efficient to develop onto. These are usually not reproducible as they might have a lot of third party dependencies such as java which are only installed into the local computer of the user and these can be inefficient to develop onto as these might need a lot manual steps in order to run the whole system. To solve this we will be using Docker and Docker Compose to build containerized pipeline.

We chose Docker as the container technology as cloud providers such as Google Cloud Platfrom and Amazon Web Services both support Docker images. [7] [12] With Docker one can run their pipeline in local or distributed environment allowing the developer to develop their system in realistic distributed virtual environment even though they would not themselves have the resources to use actual cloud service. This of course fits perfectly to the needs of our target group.

There is nothing new when it comes to containerizing a technology as this has been used as form of development for years. The problems come from the overall system. Docker images usually work very well by themselves, but when we start to integrate multiple images and technologies together the problems usually arise. This is due to the huge amount of different combination of technologies and the continous development of each which can break any of the integration with others. So each pipeline poses different problems and we try here to show what kind of problems there can be and how to solve them.

We also want the pipeline to be developed using as little different programming languages as possible. This is to take away the burden of a novice data scientist to learn multiple new languages although one can assume that this kind of person has experience from for example Python. Although Python is very popular amongst data scientists, we chose to use Scala as most of the open-source Big Data products are build on top of JVM (Java Virtual

Machine). It has also a good interoperability with Apache Spark and its programming interfaces for both object oriented and functional paradigms. On top this, Scala offers static typing which is extremely helpful to prevent errors that can occur with long computations saving developers time and resources. [48] So we think that Scala would be the most beneficial language to use in this type of scenario.

4.4 Technology selection

Next we will explain what technologies we are trying to use in each part of the pipeline. The choices are highly based on the technologies that have been seen already in use, but there are also choices that seem promising but do not have real-life examples yet. Although the end product itself is not our main objective here as we want to highlight more about the challenges and the way of building this pipeline, we still want to make the system to be as realistic as possible.

We want the system to be able to ingest normal structured stock data, but have the ability to extend to third-party metadata that can have any form and is usually unstructured. Because stock data by itself can already scale to gigabytes per day we want the ingestion have the ability to scale with input. So for the ingestion we will try to use one of the technologies that we introduced in the previous chapter: Apache Flume, Apache Kafka and Apache NiFi. All of these are scalable data ingestion frameworks that have different paradigms of handling data as seen before. For only local development these products are a bit heavy weighted, but for scalability these are necessary in order to handle massive amounts of ingested data from varying sources.

The storage should have the ability to scale to the possible terabytes of data. This means that even when amount of data is enormous, the queries should be executed in somewhat manageable time. The storage should be fault tolerant in a distributed environment in order to ensure that the data waiting to be analyzed retains its quality throughout the wait. To fulfill these requirements, we will be testing plain HDFS, Apache Cassandra and, if there is time, Apache HBase. All of these storage formats have been developed to be used in big data environment and from these HDFS and HBase were both used in some existing pipeline. HDFS does not have as good as query capabilities that could be hoped for but it offers easy integrations to other technologies and a mature development environment which is perfect for novice developers. As for Cassandra, we are trying to see whetever it is easy to integrate with the other technologies and can it bring anything to the pipeline with its high availability features.

For the analysis part of the pipeline, the pipeline should be able to preprocess the data for training algorithms that uses it and again the amount of data can be from gigabytes to terabytes. As we saw in the backgrond chapter, the current cutting edge methods for stock data analysis are deep learning methods. This is why we want the analysis frameworks have the ability to build and train these models without having to implement them from scratch.

For analysis, there does not exist that many options that work well with our requirements. The main difficulty here is to keep our pipeline using mainly Scala for programming. There are two machine learning libraries that have Scala support that they promote and can be used in big data context; Apache Spark ML and Deeplearning4j libraries which were introduced in the previous chapters. Because deep learning, and specifically LSTM networks, is the current trend in stock analysis, the library needs to have support for these. Unfortunately, Spark ML does not have these natively as the writing of this, so that leaves us with only Deeplearning4j.

Because building and demonstrating data analysis applications can be a time consuming and complex job, data science community has adopted the usage of Jupyter notebooks when implementing data analysis in Python. For the same reasons, we will be integrating Apache Zeppelin notebooks into this pipeline as these have a support for Scala programming language and allow submitting applications to Spark cluster. This is to help with reproducibility of models developed by the users.

Finally, we have the monitoring of the pipeline. We want to allow the novice data scientist to have a simple and intuitive way to manage and monitor the the state of the whole pipeline. The main goal is to give the data scientist an ability to notice errors in the pipeline as soon as possible this way preventing the errors to possibly escalate to the latter parts of the pipeline. We would also want to allow good tracking on progress of machine learning model development in the analysis stage to give the analysist tools to track the results of training.

To monitor the machine learning model development we will integrate the MLFlow tracking into the analysis phase. This is done by implementing a separate tracking server to allow this process scale separately from the actual analysis. Thus also enabling responsive tracking UI usage. If there is time, we will also try to develop a global ELK stack which was described in the previous chapter to give global monitoring on the entire pipeline.

4.5 Validation of results

The main objective of our practical research is to show that the problems described in the first chapter do exist and provide valuable information to novice data scientists on how to build reproducible and sustainable pipelines without a lot of resources. So the question is, how do we validate the results of this empirical part?

We will be using qualitive analysis focusing on reproducibility of the experiments in the pipeline, sustainability of the pipeline and how novice friendly the pipeline is. For these we will use same definitions that we have used to this point. These are not mutually exclusive terms as, for example, a pipeline that is easy to reproduce has a lot of same characteristics as novice friendly pipeline.

By reproducibility of the experiment, we refer to the amount of work and information needed to reproduce the pipeline itself and run an experiment on it. This is related to the one of the problems with many stock analysis papers that report their pipelines but from a very high level making it hard for example novice data scientists to replicate the experiments. Here, however, we will not be focusing on the amount of information published, but instead we will focus on how much manual work the user must do to run the pipeline.

We will be using the term ”sustainability” again to mean the effort needed to develop the pipeline in the future. Developing here can be divided into multiple parts, but we use it to refer the process of continuously building new features and keeping the components up to date. This is related to the problem that there is a lot documentation that shows how to get started but these instructions usually do not make the system sustainable to develop onto. We will be focusing on the aspect of how much work is needed to upgrade components in the system and how does the system support new features.

Finally, with novice friendliness we mean how much does the user has to know about underlying technologies in order to start their own experiments. We will be also using this term to analyse how much resources such as money is needed to run the experiments as this is something usually novice data scientists do not have much on their disposal.

We will be comparing the pipeline with two different pipelines using qualitive metrics. Because of the scarcity of possible points of reference the pipelines have a bit different purposes, but we are more interested how do these pipelines compare with our metrics. We have picked one pipeline from the academia and one pipeline from the industrial side. The pipelines are depicted in very high level which can make it hard to make any reasonable deductions, but given the scarcity of data available it is the best we can do. We are more interested in how they are run than the actual components that they are made of but to give a better picture we also introduce the architectures.

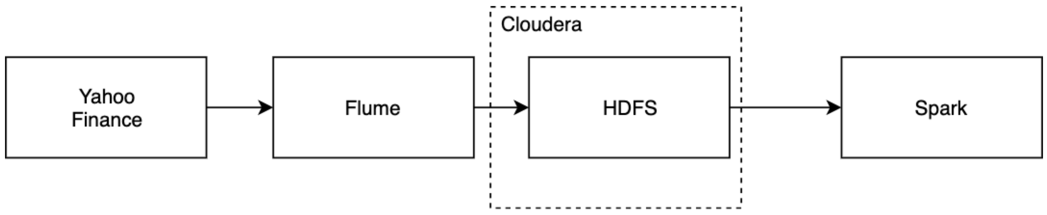

Figure 4.1: Simplified architecture of the academic pipeline

The pipeline we have picked from academia is the pipeline proposed in [50]. The reported architecture in the paper is presented in figure 4.1. From the paper we know that the system was ran locally using local Cloudera instance. Python was used with PySpark as the programming language and the pipeline ran normal machine learning models. Although this information is not much, we can use it to make some observations on our system.

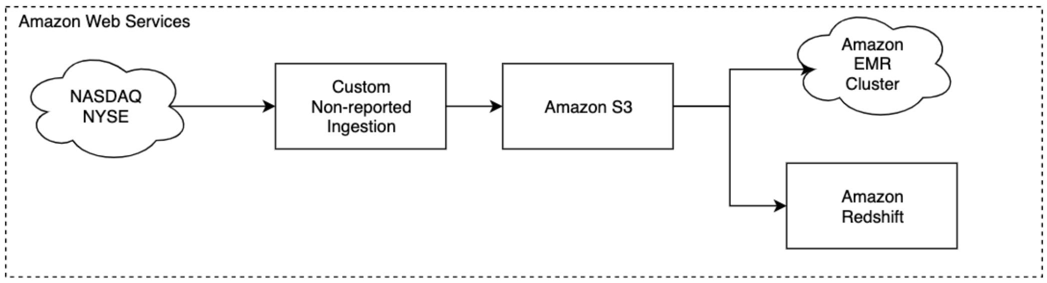

Figure 4.2: Simplified architecture of the industrial pipeline

The industrial pipeline that resembles the most of our pipeline here is the pipeline presented in [56]. The high level architecture shown is presented in figure 4.2. What can be seen in the architecture is that we do not know much about the parts of programs that analyse the data. Amazon EMR can be used to run almost many different big data frameworks but given that they report using it to analyse data it is higly likely that they are running multiple Spark clusters there. But what we do know for sure is that the system is run on Amazon Web Services in large scale and we can use this to analyse our own system from the perpectives that we listed above.

We will also be quickly reflecting on the reported challenges. We will question whetever these are significant in the sense that they could happen to any novice data scientist and is it reasonable to think of them as adversities.

We will be also analysing whetever these challenges could have been avoided and what was the root cause of these challenges.

This concludes this chapter were we examined and reasoned the plans we have for the empirical part of this thesis. In the next chapter we will report the process of the implementation and see what kind of challenges were faced during it.

Chapter 5

Implementation

In this chapter we will introduce the software created for this thesis and the challenges faced while developing it. This chapter is divided into two parts. In the first part we will examine technically what was the end result while quickly commenting on some of the choices that can be seen in the final architecture. Because of time constraints and challenges during development, there were some parts such as HBase as a storage which we could not develop. These challenges are explained in the latter part of this chapter and we continue with them in the following evaluation chapter.

5.1 Overview

In this section we will explain the practical application that was developed. We start by addressing some of the deadends that lead to the final pipeline to give reader a clearer view what changed in practice after the previous chapter. Some of the reasoning why some of the technologies were prioritized leading to not being to test every technology is presented here but more in-depth look at for specific integrations and problems will be examined in the next section. Finally in the section 5.1.3, we introduce how the pipeline can be run with any novice data scientists local machine.

5.1.1 Dead-ends at the start of development

Two technologies that were mentioned in the previous chapter were eliminated at the start of the development of each. These were Apache Flume and Apache Cassandra. As both seemed to lead to a dead-end in the development and there existed seemingly better alternatives than these, focus was moved to other technologies.

Apache Flume was supposed to be used as a component in the ingestion step, but due to its charasteristic of being mostly aimed at data which could be fetched in log form, it could not be used with the REST APIs that most of the data services use. Flume also did not seem to have any native ways to connect to storages such as Cassandra. There exists some ready-made solutions for this, but these were unmantained and seemed possible dead-ends. This is why Flume was ruled out.

Apache Cassandra was dropped from the development mostly because integrating it with other services would have required a lot of extra work. Further examination of Cassandra also showed that Cassandra is not very good choice for this specific purpose that we want to use it for. Cassandra is known for its ability to write enormous volumes of data, but when it is time to read from the storage the task becomes non-trivial. Reading from Cassandra is based on the Cassandras query language, CQL, which allows somewhat efficient queries but you have to define these high perfomance queries at the time you create tables for data. [30] Because our pipeline should be used mostly by analysist who can have varying queries into the storage this raises a challenge for future development. This is why we decided to discontinue with Casssandra.

5.1.2 Final Architecture

The final architecture can be seen in the figure 5.1. The figure presents the main dependencies between the components each arrow usually presenting the flow of data in the system with labels sometimes added to clarify better the relation. In the figure each solid rectange represents one actual server with its own process with the exception of rounded rectange to represent notebooks that are stored into the local file system. In the case of this thesis these servers were virtual machines but theoretically the system could be run with right configuration with multiple physical machines that are in the same network. Rectangles with small dashed lines such as with the spark workers represent that the server can have easily more than one instance allowing horizontal scaling. The longer dashed lines here represent bigger entities such as the HDFS cluster to make the figure more readable.

The pipeline starts from the data source that acts as the kafka producer. It reads the data from a source, in this example case from JSON files, and produces the data into a kafka topic. It also creates the needed kafka topics on startup using Kafkas admin API. Although most of the kafka documentation recommends using the command line client, this did not seem production-ready way to do this as this decouples the topic creation logic from the actual producer and would need excessive scripting in order to automate this process

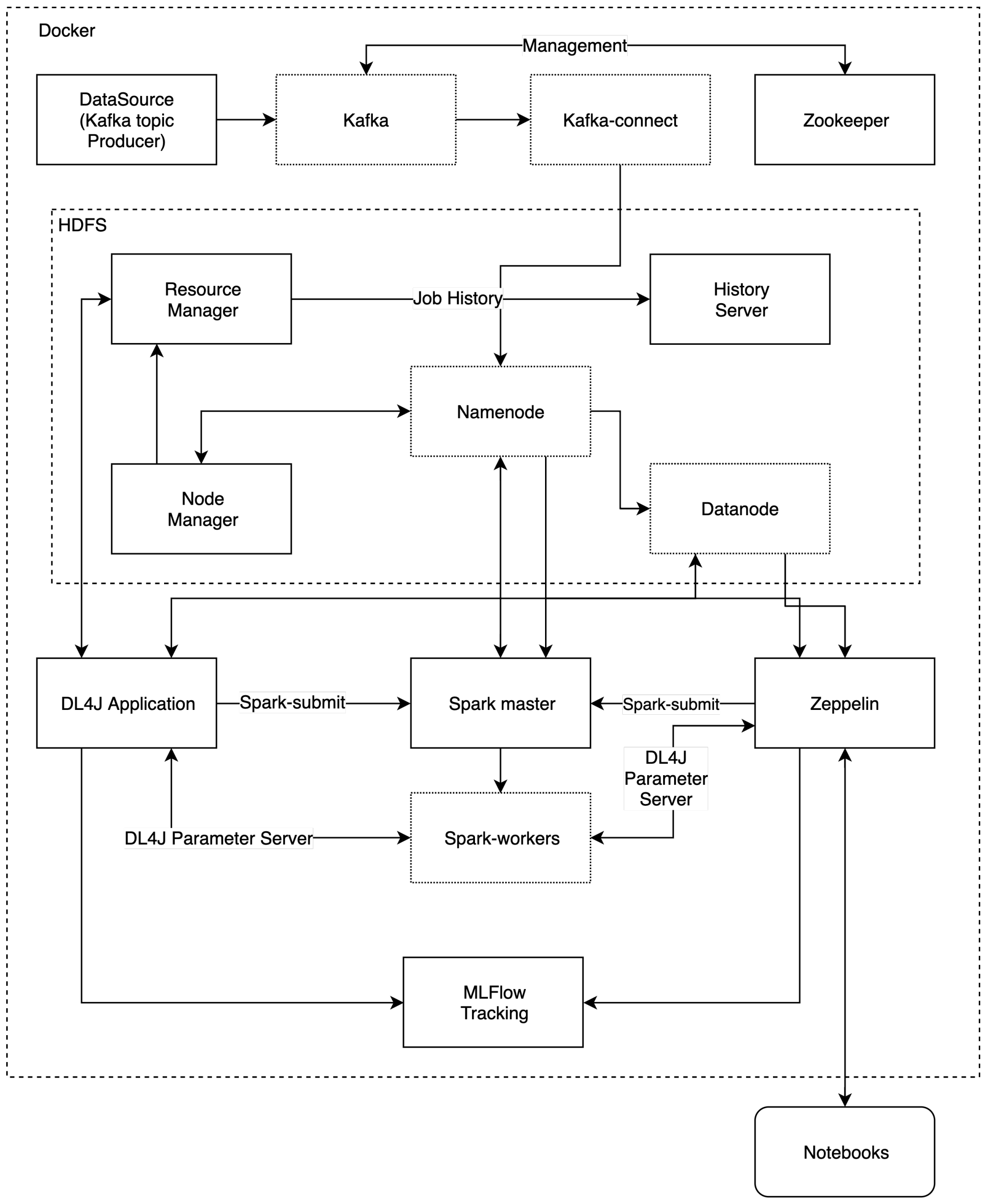

Figure 5.1: Final Architecture of the implementation section

which did not seem as maintainable as the admin API approach. This is why admin API was used in the data source server.

Then we have Kafka which is integrated into HDFS cluster with Kafka Connect HDFS 2 Sink connector which is a component made by Confluent. This acts as a kafka consumer and listens to the topic that the producer defines. Connect can be scaled and contains some basic configurations that can be used for example to define the format in which the data is stored into HDFS. We have chosen JSON as the format which the data is stored but Apache Avro is also an option. Other notable configurations that can be used to tweak the pipeline are the rate in which the topic is consumed into the HDFS and schema validation.

In the middle of the pipelin is the HDFS cluster. In this cluster there is the normal HDFS cluster with namenodes and datanodes which can be accessed to directly read the data. There is also a YARN resource management on top of this which can be used to for better manage IO routines in the cluster. This consist of the resource manager, node manager and history server nodes, that can be used as an alternative route to access data.

The HDFS cluster docker containers, as well as the containers for Spark cluster later, are a work of Big Data Europe project. The Big Data Europe is project funded by European union which one of the goals is to produce open-source tools for big data development without the need to use closed softwares.[9] As these containers were the most maintained hadoop containers at the moment of writing this, they are were the best option for this case, but they brought a couple of problems which we will examine later.

Table 5.1: Storage and ingestion components versions

	Hadoop	Kafka	Kafka Connect	Zookeeper
Version	3.1.1	2.3.0	5.2.1	3.5.5

In table 5.1 we have gathered the versions of the technologies used in the first half of the pipeline. Most are the newest versions of each software at the writing of this. The version of hadoop is specially tricky as its clients are used in multiple parts of the pipeline coupled with other software libraries which have support for only some of the older versions but do not complain if used with the newer one. This is why there can be other versions than this in the pipeline e.g in the spark cluster, but as they do not currently cause any visible errors and due to the time constraints that this project has, the versions can mismatch for now.

After the HDFS cluster comes the analysis part of the pipeline which has Spark cluster in the middle and two options to run analysis code on it. The application part allows writing production grade scala applications that can be run like any normal scala spark application. The Zeppelin is a Apache Zeppelin instance which allows writing and running notebooks that can be saved into local file system for distribution. Both approaches submit the spark application to spark using spark-submit script and the

DL4J communicates with itself with its own parameter server.

In default case, the spark application first preprocesses the json formatted data and saves this back to HDFS as csv files. In this process, it normalizes the data and appends labels to it that in our example case is just values 1 and 0 whetever the value of stock grew in n-days after the datapoint or not respectively. The data is stored back as a CSV file because DL4J is quite picky about the data format that it accepts and does not have simple default way of transforming spark DataFrames with sequential data to the Dataset format that it internally uses. After this the data goes through the training and evaluation pipeline which logs its parameters and results into MLFlow server where user can monitor the process of their different experiments.

Table 5.2: Analysis Software versions

	Spark	Scala	Java	DL4J	sbt	Zeppelin	MLFlow
Version	2.4.3	8.x	2.11.12	1.0.0-beta5	1.26	0.8.1	1.2.0

In the table 5.2 we have collected the versions of different components in this part of pipeline. Almost all of them are the latest releases of each technology at the time of implementing with the exception which is the version of Scala. Currently, the latest version of Scala is 2.13. Spark mainly supports scala 2.12, because as of 2.4.1 version of spark, the version 2.11 of scala is deprecated and there is still no support for 2.13. However, Zeppelin does not support Scala version 2.12 so the only version of Scala that still works with all of the latest releases of these two technologies is 2.11 which is why it had to be chose for the version. Java is still version 8 which is already at its end of life stage, but it was used as it is currently the safest way to ensure that your code runs correctly.

5.1.3 Usage

The initial idea of the project was to provide modular design for the pipeline where the user can cut and paste the technologies to the pipeline making it easy to produce multiple different pipelines for comparison. That is why the there is a initialization script called pipeline.sh. This is a command-line interface that asks user what technologies they want and builds a docker-compose file into build folder with all the necessary files. Sample usage is depicted in the figure 4.2. Only thing user has to do before running the script is to download manually the Kafka Connect component from confluent website and copy its content to a location noted in the project readme. This step is mandatory because unfortunately Kafka Connect did not have public distribution that would have allowed access without authentication. If this component is missing during the build, the initialization script notifies user about this before continuing further.

After the initialization, using the software is the same as with any other normal docker-compose project. Users can copy the contents of the build folder as the base of their own project or use it as it is. The project can then be build with "docker-compose build" command and run using the command "docker-compose up". After this as there will be over 10 containers running simultaneously, tools such as lazydocker are recommended for proper management.

When the containers have finally finished the startup procedures, every services user interface can be found in their default ports in localhost. The ports needed to be open to this happen, can be found in the docker-compose.yml file. To start analysing data, the user has to only open localhost:8090 where Zeppelin notebooks are running. In the project there exists an example notebook that implements a simple LSTM model. To run this example on spark cluster user has to first manually set two configurations in the spark interpreter menu that can be found under interpreter settings in the user interface. The value "master" must be set to "spark://spark-master:7077" in order to let the zeppelin use the cluster instead of local spark client. Also in the dependencies has to be added "org.apache.commons:commons-lang3:3.6" in order to have common versions in the zeppelin machine and in the cluster. The reason for the manuality of these tasks will be discussed in following sections. After this the user has to only run the cells in the notebook and the result will be saved into HDFS.

To add third party dependencies to the notebooks such as the DL4J library, the "spark.deb" syntax in notebooks is encouraged as this is the best way to preserve the dependencies for possible distribution of the notebook. This way the notebook is as independent of its environment as it can be. It is currently also the only way to preserve these if the container happens to be destroyed. Reasons for this will again be discussed in following sections.

5.2 Adversities during development

Most of the time during development went to trying to integrate these services with each other. The reason for this was mostly features that were not documented well and other misshaps such as bugs that happen when integrating two services that are not that common together. There was a lot of unique problems that did not have any documented solutions on the internet which made solving them very time consuming. In this section we will be going through the ones that took most time to solve and present the solutions at high level to these problems. More precise technical problems with their solutions can be found in this thesis' git repository that contains all the practical results.

5.2.1 Kafka-HDFS integration

Integrating Kafka topics to HDFS proved to be a non-trivial task. Unlike products like Flume, Kafka does not have any build-in sinks that would allow easy integration of different services. So the developer is left with the task of filling this hole.

The requirements for this component are also non-trivial, as it should be able to scale with the kafka and the HDFS cluster. So in order to not introduce a bottleneck to the pipeline and to do this with time limits that this thesis has, a ready-made solution was necessary. Previously, a good solution to this has been Linkedin's Camus API, but this was phased to Apache Gobblin in 2015. The problem with Apache Gobblin, however, is that it is not the lightest tool and it can move data only to Hadoop system. That is why if we were to change the HDFS storage to try some other storage option, the Gobblin should be entirely changed to something else.

Technology that solves these problems is Kafka Connect. Kafka Connect acts like a sink plugin to Kafka. Unlike with Gobblin, developers can download only the sink they actually need instead of sinks for every possible case. Because Kafka Connect has sinks for most major big data storages in the market, user is not constrained as much to Hadoop ecosystem like with the Gobblin.

The downside is that Kafka Connect is clearly Confluent product although it is open-sourced and free. Confluent has its own platform that it promotes as open-source event-streaming platform that can be used as a enterprice solution. This is why most of the Kafka Connect documentation is only about how to use it on top of this Confluent platform, although it has capabilities to run independently. This lack of good documentation introduced some obstacles for integrating this component between Kafka and HDFS especially when it came to installing and running the component.

Setting up the Kafka Connect was not that easy as it does not offer much monitoring in itself. This means that as developer you could only see if the whole system worked or did not from HDFS management console. If only some configuration was not right, Connect would only log this as an minor error in the stderr and continue running like nothing had happened although no data was flowing through it. This made debugging the connection surprisingly difficult, but after all the configurations were in place the component worked as it should throughout the rest of the development.

5.2.2 Apache Zeppelin dependency management

Major problems came when the development reached to the point of adding DL4J dependencies to the Zeppelin instance. Zeppelin has multiple ways of adding dependencies and the main way that is defined in its documentation is by its dependency UI that is located in its interpreter settings panel. This again saves these settings into interpreter.json file which is a normal JSON configuration file.

At the start of this development in order to remove the manual step of adding dependencies and other settings, we tried to save this interpreter configuration file between container restarts. This worked with every other setting except dependencies. Dependencies would conflict for unknown reasons and and the error message would change per restart. Sometimes by mere luck the dependencies would build correctly for a certain amount of time but then fail again. This would be easy to debug if Zeppelin would be a normal maven project as it is. But with ready-made containers there did not exist such an option.

Apache Zeppelin has official container which were used, but this had to be extended in order to have custom spark-submit client because the versions on spark cluster would conflict. When thought afterwards, the whole zeppelin container probably should have been build from scratch as this would have allowed better dependency management with maven.

5.2.3 DL4J sbt project

The DL4J application instance without Zeppelin is build with sbt. sbt is a build tool developed specifically for Scala applications and this why it was also chosen for this project. DL4J's main build tool that the documentation is build around is maven because it is a Java library, but it has its own sbt example in their project repository. This, however, turned out to be highly outdated and caused developing problems.

To have third-party dependencies added to a spark project there are two possible ways to implement this. First one is using the spark-submit scripts flags to define maven repositories. This was first tried but it only lead to dependency conflicts and other bugs with ivy dependency management that the script uses.

The alternative and more manageable way is to build the whole application into so called fat jar, which contains all the code to run the application including the third-party dependencies. This is what the outdated sbt example did so we opted to update this example to work with newer versions of sbt and sbt-assembly which is a plugin needed for building fat jars on sbt. To make the example project work, we migrated it from sbt version 0.X to sbt version 1.X syntax. The migration of general syntax was relatively easy, but with the newer version of DL4J library, a new merge strategy had to be written for sbt-assembly. Merge strategy is what sbt-assembly uses when it encounters duplicate files during the building process. Writing a updated merge strategy for DL4J was not a trivial task as the strategy that was usually suggested did not work in this case and lead to errors in runtime. After quite a bit of trial and error, we thankfully managed to create a merge strategy that did not throw away necessary files needed at the runtime.

5.2.4 Running the application in Spark

Now we move on to the problems that were really hard to debug when the problems in the previous two subsections were persisting at the same time. When running a Zeppelin task on Spark and the program receives error, Zeppelin correctly logs the stack trace from Spark workers stderr. What it does not do, is logging the the whole stack trace but only the two to three first errors. This made it seem like the error was a some upper level error when instead the actual error that happened in lower part of the stack. This was resolved by noticing the full stack when the application was run on application server.

After this, the next problem was that Intels mkl-dnn library did not seem to be in the classpath of DL4J subdependency library javacpp. This is the code that is responsible of allowing the java code to use high efficiency feature of C++ language. This of course did not have any clear solutions online except some vague conversation logs about possibility of glibc missing in the parent OS and by accident this turned out to be the exact problem in this case too. The Spark worker docker container, which was a result of the Big Data Europe project, was build on top of alpine linux image which has phased out to use musl libc library instead of glibc. There exists tools that can be used to install glibc to alpine instance and we tried this, but after multiple errors in multiple different attempts to install the library, we opted out to fork the docker image to use debian based image. This resolved all the problems that had been until this point and the application finally compiled in worker machine.

The problems after this did not cause the application process on worker to crash but were logged into worker machine while the process continued to run. These were somewhat minor but laborous problems such as allocating enough disk for /dev/smh shared memory implementation and opening and defining right ports for DL4Js parameter server that handled the parameter update logic. Once all of these were fixed, the next problem was that the training process in spark did not finish at all and seemed to go into a state of infinite loop without giving any errors on the driver logs. This seemed like a dead-end until we noticed that a spark job before this had silently failed. It seems that the DL4Js training code was beforehand trying to temporarily save the data into hdfs and this failed as something went wrong. This step in the process could be skipped with configurations and after this the model training process finally succeeded.

The final problem we faced was a randomized index out bounds exception. Raising the number of epochs we noticed that this occured at random, and unlike the previous errors, this seemed like a error in the library. The code that raises this error handles parallel code so if we were to make conjectures from this it would seem like there would be an error with shared memory access in some part of the asynchronous code. However at this point, the time allocated to this project was already running out so we could not confirm this definitely.

Now that we have seen what we were able to implement during this thesis we move on to evaluate the quality of this software. We will also discuss more about what went well, what went wrong and what could have been done better.

Chapter 6

Discussion

In this final chapter before conclusion, we will be validating that the pipeline that was build in the previous chapter is more useful to novice data scientists than the ones that are somewhat badly reported. We will be using the qualitive metrics that we defined in the planning chapter before implementation. We will end this chapter by contemplating on how this pipeline and process could improved in the future.

6.1 Method validation

In this section we will be focusing on the three metrics we introduced previously. We will be comparing our way of building with the two existing pipelines to see which are the advantages and disadvantages of using one for novice data scientist.

6.1.1 Reproducibility

As stated before, in this characteristic we will be focusing on how much manual work must the user do in order to run the pipeline. Using docker and docker compose to build the pipeline we were able to limit the number of external dependencies to only these two when a new user wants to run the pipeline and with only couple minor exceptions we were able to automate the local setting up process. Also the usage of zeppelin notebooks allows the user to share their models with anybody who has the same pipeline or other means to run them.

When we compare this to the academic pipeline which had all of its components as local dependencies, the amount of possible work is greatly smaller as the user does not have to install each component to their local machine and worry about their versions. As what comes to the automation of the setting up the pipeline, we have no information on this to make any reasonable claims.

When compared to the industrial pipeline, although not reported, we can pretty much assume that the industrial pipeline is highly reproducible as there are probably ten to hundred employees working with the pipeline and in this sense is probably a lot more reproducible than the pipeline presented here. This could be due to inherent nature of containarized applications in AWS and the tooling that continuous integration tools and Amazon provide.

6.1.2 Sustainability

Compared to the local academic pipeline, where each components version has to be manually confirmed when it is installed and updating components means that everyone who wants to run newer versions of the pipeline has to do the same, our version does not suffer from these problems as these are defined in the docker configuration files. Compared to the industrial pipeline, however, the process is probably taken into step further as amazon can keep the components better up to date by automatically updating minor versions, which our current methods do not necessarily do. This depends a lot on the docker image provider and their methology on versioning.

6.1.3 Novice friendliness

With our choices on docker and scala we have achieved somewhat easier pipeline to handle for novices. With Docker it was possible to abstract away a lot of the complexity which there was to run the different components. This of course increased the amount of work to build the pipeline but once it was built it could be run without any knowledge on underlying system.

The choice of Scala was reasoned because of its interoperativity with spark and support for multiple paradigms. It also ran over JVM which made it easier to justify it as all the other technologies ran too. However as seen with the challenges in the previous chapter, somewhat better choice for novices could have been Python. Both of these languages are taught as beginner languages in Aalto University, but Python might have had better support for deep learning models although integrating this with Spark would have probably still introduced challenges.

When compared to the academic pipeline, the amount of work to build this first time, was probably greater than with the academic pipeline because Docker brought one more component to integrate with every other component. This could be seen in the challenges faced that we examined in the previous chapter.

Compared to the industrial pipeline, with the help of docker we were able to make the pipeline easy to run locally in a virtually distributed setting without spending any money. This can be very crucial difference because as stated before, our target group definitely does not have as much resources than the company running the reference industrial pipeline.

6.2 Reflecting challenges

Next we question, whetever the challenges faced during development were significant in the sense that they could actually cause harm to novice data scientists.

With the Kafka-HDFS integration we can argue that the necessary information was available online and anyone looking to build the integration could have found this with little to none research, which is absolutely correct. From the ocean of possible alternatives, we could have found other solutions also such as Flume which some were suggesting. This is not the worst kind of problem there is as the information, although scattered, is available in the internet. The actual problem here is that because there is no clear consensus on what to use and what are the advantages and disadvantages over another, the process of picking right technology can be very stressful. What can be said about implementing this integration itself is that we can critize our solution to run it outside Confluent environment making the environment itself non-friendly for novices, in which case the lack of documentation is only a problem caused by ourselves to ourselves, which is true.

With the Zeppelin dependency management problems the real problems started. Instead of giving easy to understand way to handle the dependency errors, user is mostly left with learning maven dependency management which is hidden inside the zeppelin itself. Probably the best solution would have been to build the service on top of docker from ground up. This, however, is far from novice friendly way to use the component as it needs a lot of external knowledge to produce a valid setup which is why the usage of the software itself becomes very hard for novices.

The DL4J with Scala using sbt, the major thing here to be questioned is the choice of sbt. As there was no up to date DL4J project with sbt we can make a conclusion that sbt is not relevant in this field. sbt, however, is the most well known build tool for Scala so if a library has a scala support it would not be far fetched to hope a working up to date example of this as most novices really do need them as reference to build their own applications.

Finally we had the problem with running the DL4J on a docker based

Spark instance with Zeppelin. This as a combination is definitely not the combination that probably any of the singular component makers probably thought which is reasonable as there are so many possible options. This highlights well the challenges that come from having so many different possible options that are recommended by themselves without context. One can question whetever we did caused this to ourselves and as the stack can be thought very exotic the blame is all ours in this sense, but given the reasoning we used for each choice in this stack, we think that is reasonable to assume that this could happen anyone with the same kind of requirements. Of course a lot of this could have been avoided if Scala was replaced for example with Python but these are assumptions we cannot prove to be wrong or right.

6.3 Future improvements

The lack of reference points made it hard to compare this way of building a pipeline with other possible ways. Hopefully in the future, the information about these systems would become more open so that we could better understand them, but until then we have to work with what we have.

Other improvements that could have been done is that the model training could have been configured to run on GPU which is the current de facto way to train machine learning models. Unfortunately due to lack of time this was not possible, but would be a great way to improve the information here as there is not that many examples online of this kind of process.

Chapter 7

Conclusions

In this thesis, we examined big data pipelines to analyze stock market data from the point of view of a novice data scientist. We examined what are the challenges that novice data scientists face when starting their development. This was examined from both the stock analysis side and the big data technology side. Here we saw that the available practical information on stock analysis is very limited. In the big data technology side the problems were mostly the lack of intermediate information or the fact that it is very scattered.

We conducted literary research on current trends on methods that are used to analyze stock market data. From this we saw that deep learning models are currently the driving force in stock market analysis. Other statistic methods are also used with conjuction of data from other varying sources. These methods are not only used to predict prices in hopes for profit, but can also be used to analyze e.g causalities in other phenomenons.

We researched what are the technologies currently used in pipelines that analyze stock market data covering both academic and industrial use and saw that public information about these is quite limited. With the information publicly available, we saw that usually only the analysis phase was reported and other aspects of the systems were dismissed. These other aspects were most of the time monitoring of the system, but also ingestion and storage were not reported in many cases.

We planned and implemented a basic stock data analysis pipeline based on the results from the previous literary studies and technologies that could help novice data scientists to develop their systems in the future. Our goal was to bring down the gap between beginner level documentation and corporate level complex systems and highlight the problems that novice data scientists could face while developing such a system. We were able to build such a pipeline and report multiple significant challenges while developing the pipeline which could affect our target group. We validated the pipeline using 3 different qualitive metrics and compared the results with 2 different existing pipelines. We saw that each one had its advantages and disadvantages depending on the metric. We also questioned the problems we faced whetever these were something that novice data scientists could face and saw that this was highly likely.

In the future, hopefully, more information can come available if and if these technologies gain popularity and the role of big data grows. Until then we have to cope with the information available and just try to produce more information to make the field more accessible for new novice data scientists. This thesis hopefully alleviates someones process, in the field of stock data analysis.

Bibliography

[1] Alpha vantage documentation. https://www.alphavantage.co/ Accessed 23.09.19.

[2] Apache flume documentation. https://flume.apache.org/ Accessed: 16.06.19.

[3] Apache kafka documentation. https://kafka.apache.org/ Accessed: 16.06.19.

[4] Apache nifi documentation. https://nifi.apache.org/docs.html Accessed: 20.06.19.

[5] Apache spark documentation. https://spark.apache.org/docs/latest/index.html Accessed: 26.06.19.

[6] Apache systemml documentation. http://apache.github.io/systemml/index.html Accessed 30.06.19.

[7] Aws: How to deploy docker containers. https://aws.amazon.com/getting-started/tutorials/deploy-docker-containers/ Accessed: 23.09.19.

[8] Barchart free market apis. https://www.barchart.com/ondemand/free-market-data-api Accessed 23.09.19.

[9] Big data europe project. https://www.big-data-europe.eu/ Accessed 25.09.19.

[10] Deeplearning4j documentation. https://deeplearning4j.org/docs/latest/ Accessed: 30.06.19.

[11] Elastic stack documentation. https://www.elastic.co/elk-stack Accessed: 30.06.19.

[12] Gcp: Quickstart for docker. https://cloud.google.com/cloud-build/docs/quickstart-docker Accessed: 23.09.19.

[13] Hdfs documentation. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html Accessed: 22.06.19.

[14] Iex trading documentation. `https://iextrading.com/` Accessed 15.04.19.

[15] Intrino documentation. https://intrinio.com/ Accessed 23.09.19.

[16] Metastock homepage. https://www.metastock.com/ Accessed 01.12.19.

[17] Mlflow documentation. https://mlflow.org/docs/latest/index.html Accessed: 30.06.19.

[18] Quandl us eod data documentation. https://www.quandl.com/data/EOD-End-of-Day-US-Stock-Prices Accessed 23.09.19.

[19] T2000 homepage. https://www.worden.com/ Accessed 01.12.19.

[20] World trading data documentation. https://www.worldtradingdata.com/ Accessed 23.09.19.

[21] Amirghodsi, S., Alla, S., Karim, M. R., and Kienzler, R. *Apache Spark 2: Data Processing and Real-Time Analytics.* Packt Publishing, 2018.

[22] Bartram, S. M., and Grinblatt, M. Agnostic Fundamental Analysis Works. *Journal of Financial Economics (JRE)* (June 20 2017).

[23] Bridgwater, A. Nsa 'nifi' big data automation project out in the open. https://www.forbes.com/sites/adrianbridgwater/2015/07/21/nsa-nifi-big-data-automation-project-out-in-the-open/#79a9319655d6 Accessed: 20.06.19.

[24] Chen, J. *Essentials of Technical Analysis for Financial Markets.* John Wiley & Sons Inc, 2010.

[25] Chen, L., and Yang, C.-Y. Stock price prediction via financial news sentiment analysis. https://github.com/Finance-And-ML/US-Stock-Prediction-Using-ML-And-Spark Accessed: 08.05.19.

[26] Cheng, J. Real time machine learning architecture and sentiment analysis applied to finance. Slide set available at: `https://www.slideshare.net/Quantopian/real-time-machine-learning-architecture-and-sentiment-analysis-applied-to-finance` Accessed: 08.05.19.

[27] Das, S., Behera, R. K., Kumar, M., and Rath, S. K. Real-time sentiment analysis of twitter streaming data for stock prediction. *International Conference on Computational Intelligence and Data Science* (2018).

[28] David Andrešić and Petr Šaloun and Ioannis Anagnostopoulos. Efficient big data analysis on a single machine using apache spark and self-organizing map libraries. *12th International Workshop on Semantic and Social Media Adaptation and Personalization* (2017).

[29] Day, M.-Y., Ni, Y., and Huang, P. Trading as sharp movements in oil prices and technical trading signals emitted with big data concerns. *Physica A: Statistical Mechanics and its Applications 525* (2019).

[30] Dronavalli, A. Why apache cassandra is not good for timeseries data & analytics. https://medium.com/@abhidrona/why-apache-cassandra-is-not-good-for-timeseries-data-analytics-a1d65a369048 Accessed 24.09.19.

[31] Fischer, T., and Krauss, C. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research* (2017).

[32] Fox, J. *Myth of the Rational Market.* Harper Business, 2009.

[33] George, L. *HBase: The Definitive Guide, 2nd Edition.* Packt Publishing, 2018.

[34] Gu, R., Zhou, Y., Wang, Z., Yuan, C., and Huang, Y. Penguin: Efficient query-based framework for replaying large scale historical data. *IEEE Transactions on parallel and distributed systems 29* (2018).

[35] Hoffman, S. *Apache Flume: Distributed Log Collection for Hadoop - Second Edition.* Packt Publishing, 2015.

[36] Islam, S. R., Ghafoor, S. K., and Eberle, W. Mining illegal insider trading of stocks: A proactive approach. *IEEE International Conference on Big Data* (2018).

[37] JOHN L. PERSON. *Mastering the Stock Market: High Probability Market Timing and Stock Selection Tools.* John Wiley & Sons, Incorporated, 2013.

[38] KHASHEI, M., AND HAJIRAHIMI, Z. A comparative study of series arima/mlp hybrid models for stock price forecasting. *Communications in Statistics Simulation and Computation* (2018).

[39] KUMAR, H., AND KUMAR, M. P. Apache storm vs spark streaming. `https://www.ericsson.com/en/blog/2015/7/apache-storm-vs-spark-streaming` Accessed: 19.05.19.

[40] KYO, K. Big data analysis of the dynamic effects of business cycles on stock prices in japan. *15th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN)* (2018).

[41] LE, L., AND XIE, Y. Recurrent embedding kernel for predicting stock daily direction. *IEEE/ACM 5th International Conference on Big Data Computing Applications and Technologies* (2018).

[42] LEE, C., AND PAIK, I. Stock market analysis from twitter and news based on streaming big data infrastructure. *IEEE 8th International Conference on Awareness Science and Technology* (2017).

[43] LOTTER, J. C. Bye yahoo, and thanks for all the fish. `https://financial-hacker.com/bye-yahoo-and-thank-you-for-the-fish/` Accessed: 19.05.19.

[44] MUN, F. W. Big data, small pickings: Predicting the stock market with google trends. *The Journal of Index Investing 7* (2017).

[45] MURPHY, J. J. *Technical analysis financial markets: A comprehensive guide to trading methods and applications.* New York Institute of Finance, 1999.

[46] NAM, K., AND SEONG, N. Financial news-based stock movement prediction using causality analysis of influence in the korean stock market. *Decision Support Systems 117* (2019).

[47] NEERAJ, N., MALEPATI, T., AND PLOETZ, A. *Mastering Apache Cassandra 3.x - Third Edition.* Packt Publishing, 2018.

[48] NICOLAS, P. R., MANIVANNAN, A., AND BUGNION, P. Scala: Guide for data science professionals, 2017.

[49] Palmer, N., Ricker, T., and Page, C. DATA & ANALYTICS: Analyzing 25 billion stock market events in an hour with NoOps on GCP. Available at: `https://www.youtube.com/watch?v=fqOpaCS117Q` Accessed: 08.05.19.

[50] Peng, Z. Stocks analysis and prediction using big data analytics. *International Conference on Intelligent Transportation, Big Data & Smart City* (2019).

[51] Rao, T. R., Mitra, P., Bhatt, R., and Goswami, A. The big data system, components, tools, and technologies: a survey. *Knowledge and Information Systems* (2018).

[52] Senguptaa, S., Basaka, S., Saikiab, P., Paulc, S., Tsalavoutisd, V., Atiahe, F., Ravif, V., and Peters, A. A review of deep learning with special emphasis on architectures, applications and recent trends.

[53] Sezer, O. B., Ozbayoglu, A. M., and Dogdu, E. An artificial neural network-based stock trading system using technical analysis and big data framework.

[54] Sezer, O. B., Ozbayoglu, A. M., and Dogdu, E. A deep neural-network based stock trading system based on evolutionary optimized technical analysis parameters. *Procedia Computer Science 114* (2018).

[55] Skuza Michal and Romanowski Andrzej. Sentiment analysis of twitter data within big data distributed environment for stock prediction. *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)* (9 2015).

[56] Snively, B. AWS re:Invent 2018: Big Data Analytics Architectural Patterns & Best Practices. Available at: `https://youtu.be/ovPheIbY7U8` Accessed: 08.05.19.

[57] Utthammajai, K., and Leesutthipornchai, P. Association mining on stock index indicators. *International Journal of Computer and Communication Engineering 4* (2015).

[58] Vartak, M., Subramanyam, H., Lee, W.-E., Viswanathan, S., Husnoo, S., Madden, S., and Zaharia, M. Modeldb: A system for machine learning model management.

[59] Voulgaris, Z. *Data Scientist: The Definitive Guide to Becoming a Data Scientist.* Technics Publications, 2015.

[60] Wang, W. A big data framework for stock price forecasting using fuzzy time series. *Multimedia Tools and Applications 77* (2018).

[61] World Federation of Exchanges. Monthly report january 2019. https://www.world-exchanges.org/our-work/statistics Accessed 15.04.19.

[62] Yarabarla, S. *Learning Apache Cassandra - Second Edition.* Packt Publishing, 2017.

[63] Yu-Cheng, K., Jonchi, S., and Jim-Yuh, H. eWOM for Stock Market by Big Data Methods. *Journal of Accounting, Finance & Management Strategy 10* (2015).

[64] Yuan, D. Stream processing in uber. https://www.infoq.com/presentations/uber-stream-processing/ Accessed: 19.06.19.