UiT THE ARCTIC UNIVERSITY OF NORWAY

The Faculty of Science and Technology
Department of Computer Science

Limelight: Real-Time Detection of Pump-and-Dump Events on Cryptocurrency Exchanges Using Deep Learning

—

Andreas Isnes Nilsen
INF-3981: Master's Thesis In Computer Science
June 1st 2019

"Long cat, is long." – Robert Pettersen

"Fått laksekrem på hjernen." – Maren Sofie Ringsby

"Reality continues to ruin my life." – Bill Watterson

Abstract

Following the birth of cryptocurrencies back in 2008, internet investment platforms called exchanges were created to constellate these cryptocurrencies. Allowing investors to sell and buy assets equitable and agile over a single interface. Exchanges now have become popular and carry out over 99% of all daily transactions, totaling hundreds of millions of dollars. Despite that exchanges handling enormous quantities of money, the industry remains mostly unregulated.

As long as these exchanges remain unregulated, they are and will continue to be susceptible to price manipulation schemes since they are legal to perform by law. Over the years, exchanges have grown into an attractive field where scammers execute various frauds that aims to leech assets from ordinary investors. One particular scheme has risen in popularity over the years and often observed at exchanges, and that is Pump-and-Dump. This scheme has a history from all the way back in 1700 and is still active and troublesome for investors today.

In this thesis, we present Limelight, a system that seeks to detect Pump-and-Dumps in real-time using deep learning. Throughout this thesis, we retrieved, prepared, labeled, and processed a dataset to train a model that identifies Pump-and-Dumps. With high accuracy, the model surpasses previously proposed models in the detection of Pump-and-Dumps.

Acknowledgements

First and foremost, I would like to thank my supervisor Robert Pettersen for being awesome. As throughout this last year, your ears had to endure severe pain from my twaddling gibberish-squawking beak. And yet, you managed my fuss until the very end without taking me to the vet and putting me down. I appreciate that you invested so much of your time to help and guide me, but also *whip crack* "encouraging" me to work.

I also want to thank the lords and ladies in my class for making my time as a student... just awesome, cool. I don't know. One thing is for sure, I will never forget all those ridiculous late night hours on the university we have had. There are especially two fellow students I want to express my gratitude to, Sverre Coucheron(ny) and Nikolai Magnussen. You guys, are, just, ok? Just kidding, love you guys!

Lastly, I want to thank Ingrid Tiedemann Gjersøe for all the cozy weekends with beverages and delicious food. It is surely good to occasionally substitute my everyday Grandiosa with actually nutritious fodder.

Contents

List of Figures

List of Tables

List of Code Listings

List of Equations

1 Introduction

The value of cryptocurrencies has rapidly increased on the last years. In 2018 cryptocurrencies had a market capitalization of around $300 billion according to CoinMarketCap, making it comparable to Denmark's Gross Domestic Product (GDP)[1]. Despite the high market capitalization, these cryptocurrencies are mostly unregulated, including the investment platforms called exchanges where investors trade cryptocurrencies and fiat money[1]. Due to the anonymity and lack of regulation, this ecosystem has become an appealing field for conducting illegal activities like terrorism, money laundering, customer theft, and fraud [3].

Exchanges play a central role as they are popular among investors and carry out 99% of all cryptocurrency transactions [4]. Unsurprisingly that makes them vulnerable to scammers who seek to pray on the misinformed [1]. One particular scam that has become popular in cryptocurrency markets over the last few years is the price manipulation scheme Pump-and-Dump (P&D) [5]. P&D involves artificially inflating the price of a cheap asset (pump) on an exchange and selling the purchased assets at a higher price. Once the assets are sold off, the price falls (dump) and the affected investors lose their money to those who organized the scam [6]. Two researchers at the Imperial College London revealed that at least two P&D schemes are executed daily on a cryptocurrency market, producing roughly $7 million in daily trading volume [5].

1. Money made by the government[2]

As these scammers corrupt exchanges and deceive investors, people are now reluctant to invest in cryptocurrencies due to mistrust and scepticism [7]. In the last two years, a few articles [1, 5, 6, 8] have proposed various methods for detecting P&Ds, but none have yet proposed a model that detects P&Ds in real-time using deep learning. Detecting P&Ds in real-time allows unethical investors to improve upon their trading strategies by having the opportunity of participating in P&Ds. But it also allows exchanges to prevent P&Ds, making them more trustworthy.

The incentive of using deep learning is primarily because of the tremendous amount of data cryptocurrency sources continuously produce. Machine Learning (ML) is generally good at solving problems that have large-scale datasets [9]. Secondly, detecting P&Ds using a rule-based solution is tricky with high-dimensional data, while deep learning has turned out to be very good at it [10]. Third, deep learning completely outperforms traditional ML methods as the scale of data increases [11, 12].

In this thesis, we present Limelight, a system that seeks to detect P&Ds in real-time using deep learning. Limelight retrieves, and stores live data seamlessly from multiple cryptocurrency sources. The gathered raw data flows through several enrichment stages to prepare the data for ML. In every supervised learning problem, training a model requires prior knowledge of each sample, and because of the infeasibility of manually collecting P&Ds, it uses an anomaly detection algorithm to pinpoint suspicious time intervals in historical data that may contain P&Ds. To reduce the number of false anomalies, the alleged anomalies goes through a manual filtering process. With both data and labels, we define a labeled dataset to train a model that contains a network of connected serialized layers where each layer incorporates a ML.

1.1 Problem Definition

A popular price manipulation scheme carried out on cryptocurrency exchanges is P&Ds. We investigate if we can construct a system, named Limelight, that detects these in real-time using deep learning. Thus, our thesis statement is:

> *Real-time classification of Pump-and-Dumps (P&Ds) in cryptocurrencies can be done using deep learning.*

1.2 Methodology

The final report of the Task Force on the Core of Computer Science[13] define the succeeding three major paradigms as the discipline of computing. The first paradigm, *theory*, is rooted in mathematics and consists of four steps followed in the development of a coherent, valid theory:

1. characterize objects of study (definition);
2. hypothesize possible relationships among them (theorem);
3. determine whether the relationships are true (proof);
4. interpret results.

The second paradigm, *abstraction* (modeling), is rooted in the experimental scientific method and consists of four stages that are followed in the investigation of a phenomenon:

1. form a hypothesis;
2. construct a model and make a prediction;
3. design an experiment and collect data;
4. analyze results.

The third paradigm, *design*, is rooted in engineering and consists of four steps followed in the construction of a system (or device) to solve a given problem:

1. state requirements;
2. state specifications;
3. design and implement the system;
4. test the system.

This thesis adheres to the paradigms abstraction and design. We investigate our thesis statement's viability through constructing, experimenting, and analyzing. By means of constructing, we design and implement a system Limelight, that pursue to solve the stated problem. State requirements and specifications advance and change throughout this thesis by experimenting with various designs. Testing the system involves analyzing Limelight's abilities.

1.3 Context

This thesis is written in the context of Corpore Sano[2], a center that conducts joint research in the fields sports, medicine, and computer science. Our interdisciplinary research targets elite sports performance development and injury prevention; preventive health care; large-scale population screen; and epidemiological health studies. In the field of computer science, we have a focus on Research and Development (R&D) systems for monitorization, back-end storage, ML, and analytics.

Two of our projects involves injury prevention and performance development for the elite soccer players in Tromsø IL (TIL), our tightly partnered club. With the systems Bagadus [14] and Muithu [15] currently deployed and used at TILs practices and games. Muithu is a portable video annotation system that integrates real-time coach notations with related video sequences. While Bagadus is a real-time prototype of a sports analytics application, it integrates a sensor system, a soccer analytics annotations system, and a video processing system using a video camera array. A prototype is currently deployed at Alfheim Stadium in Norway, TILs home ground.

In the wake of the interest in cryptocurrencies and blockchain technology, Corpore Sano did a longitudinal study of this ecosystem's most prominent cryptocurrency, Bitcoin [16]. The study investigated how the scalability affects the performance, and how the costs and fees are dependent. The study also proposed two machine learning models that can predict the bandwidth of scheduled transactions according to the fee payers are willing to offer, and the expected revenue for miners according to the time spent mining.

Another blockchain related contribution from Corpora Sano is FireChain [17]. It combines a byzantine fault-tolerant gossip service and full membership, with a proposal for blockchain systems that does not consume excessive energy. This protocol is building upon FireFlies [18], an overlay network protocol. The results show that FireChain is feasible, scalable, and use less power than other blockchain related consensus protocols.

2. http://www.corporesano.no/

1.4 Outline

Chapter 2 first describes three well-known software architectures we use throughout this thesis. Then it describes cryptocurrencies and their trading platforms, exchanges, and details about P&Ds. Then, it briefly describes deep learning and the structure of an artificial neural network. Finally, it describes some related work in the detection of P&Ds.

Chapter 3 presents the overview of Limelight, and details each component, and Limelight's different phases.

Chapter 4 covers the implementation of Limelight.

Chapter 5 evaluates Limelight's prediction abilities.

Chapter 6 summarizes this thesis, presents Limelight's results and contributions, and outlines future work.

2 Background and Related Work

This chapter presents a theoretical background in various fields that are relevant throughout this thesis. The first section describes the software architectures we use during this thesis. The second section contains information regarding cryptocurrencies, and their exchanges, and how Pump-and-Dump (P&D) organizers execute their scheme on these exchanges. The third section covers an introduction to deep learning. Finally, the last section presents related work in the detection of P&Ds.

2.1 Software Architectures

A software architecture define the structure and relation between components within a system, and it is developed as the first step toward designing a system that has a collection of desired properties. An architectural view is abstract, distilling away details of implementation, algorithm, and data representation and concentrating on the behaviour and interaction of elements [19, p. 3].

2.1.1 Master/slave

The term master/slave is through a quiet ongoing debate in the coding community [20–22], as some may interpret it offensively and relate to the institution of slavery. Our intention is not to insult anyone, but we are still using this term because none have yet proposed a lasting substitution to it.

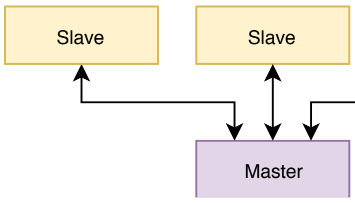


Figure 2.1: master/slave architecture. The master assigns tasks to the slaves. Soon as the slaves complete their task, they either send the result to the master, or the master gathers the result from them.

Nevertheless, In computer science, master/slave is a model for a communication protocol in which one process (master) controls one or more processes (slaves) [23]. Figure 2.1 is an example that illustrates such a hierarchy with a master and its three slaves. A typical design pattern in parallel computation is where each slave is assigned a computational task from the master, and they make the computation accordingly, then the master gathers the result from them. This procedure continues as long as it is necessary. Some of the advantages and disadvantages of this software architecture are:

Advantages

1. *Synchronization pitfalls* - the master is a single sequential process that synchronizes with the slaves, which makes handling of typical pitfalls like race conditions and deadlocks uncomplicated.

2. *Horizontal scaling* - the system scales by adding slave instances [24]; there is no upper limit of them. As slaves work in isolation without synchronization, the bottleneck, is the master if it has too many slaves to communicate with.

3. *Faulty slave tolerance* - if a slave crash, the system will continue to work, but it will also result in partially executed work.

Disadvantages

1. *Single Point of Failure (SPOF)* - if the master crash, the whole system will face downtime.

2.1.2 Publish/subscribe

The Publish/Subscribe (Pub/Sub) interaction paradigm provides subscribers with the ability to express their interest in an event or a pattern of events, in order to be notified subsequently of any event. The publishers generate events that match their registered interest. There are variations of these Pub/Sub systems such as *topic-based*, *content-based*, and *type-based* [25], and it is infeasible to cover all of them, but, these systems do share some basic commonalities which we will describe.

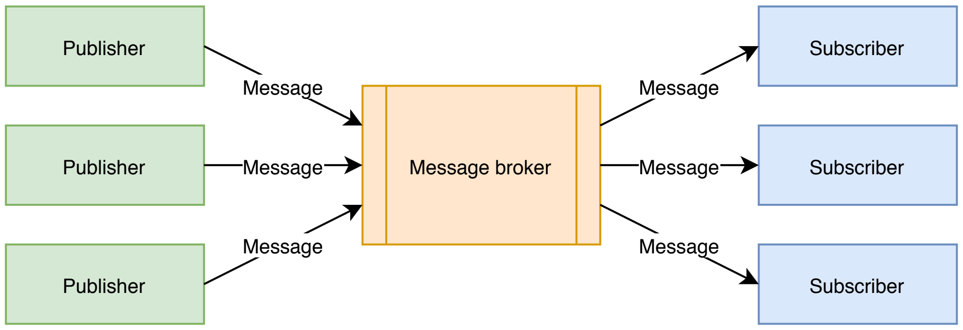


Figure 2.2: Publish/Subscribe architecture. The publisher sends events/messages to the broker. The subscribers can expresses interest in specific events to the broker. The broker propagates publishers events to these subscribers that have previously expressed interest in said events.

In a basic Pub/Sub system (Figure 2.2), there are publishers, subscribers, and a message broker. Where the publishers send messages containing some information to the message broker's event service system. The subscribers can announce interest to the broker in which messages they want to receive. Once the broker receives a message, it gets propagated to the subscribers that have expressed interest in receiving those messages. The subscribers can also unsubscribe.

According to Amazon Web Services (AWS) [26] and CodeKraft [27], the advantages and disadvantages with Pub/Sub systems are:

Advantages

- *Eliminate polling* - the flow of messages provides significant advantages to developers who build applications that rely on real-time events. Messages allow instant, push-based delivery, eliminating the need for message consumers to poll for new information and updates periodically.

- *Dynamic targeting* - the organization of data is more natural and less error-prone. Instead of maintaining a roster of peers that an application can send messages to, a publisher will post messages. Then, any interested party will subscribe to its endpoint and start receiving these messages.

- *Decouple and scale independently* - makes the software more flexible. Publishers and subscribers are decoupled and work independently from each other, which allows scaling them independently.

- *Simplify communication* - communication and integration code is some of the hardest code to write. The Pub/Sub model reduces complexity by removing all the point-to-point connections with a single connection, which will manage subscriptions to decide what messages should be delivered to which endpoints.

Disadvantages

- *Reliability* - the broker might not notify the system of message delivery status; so there is no way to know of failed or successful deliveries. Tighter coupling is needed to guarantee this.

- *Decouple and independence* - despite that decoupling and independence is an advantage, it can also be a disadvantage. Updating relationships between subscribers and publishers can be a thorny issue, and publishers do not know the status of the subscriber and vice versa.

- *Broker dependence* - the need for a broker, message specification, and participant rules add some more complexity to the system.

2.1.3 Pipeline

The pipeline architecture has received considerable attention since 1960, and it is a form of embedding parallelism or concurrency into a system [28]. The pipelining concept is also found in Central Processing Unit (CPU) architectures to accelerate the execution of instructions. One can interpret a pipeline (Figure 2.3) as a processing bus with several stages running in parallel, where the stage's output is the next stage's input. The final stage's output is the whole pipeline's output.

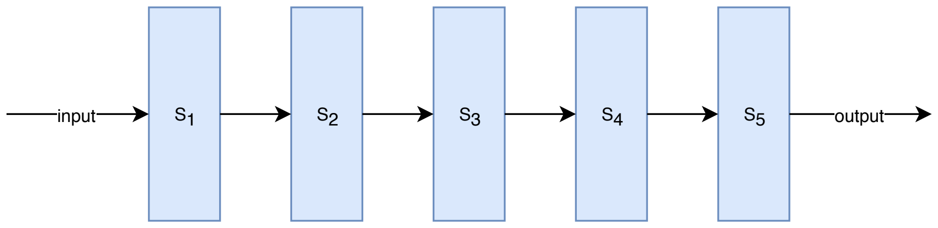


Figure 2.3: pipeline architecture, where each stage output is the succeeding stage's input. Every stage executes in parallel.

According to [29], some advantages with pipelines are:

- *Flexibility* - computational stages are easy to replace.
- *Extensibility* - the system is partitioned into components, which makes it easy to create new functionality.
- *Scalability* - each part of the computation is presented via a standard interface. If any part of the pipeline have an performance issue, it is possible to scale each component independently.

2.2 Cryptocurrency

Cryptocurrencies are digital or virtual assets, and they use cryptography as a security and consistency mechanism [1, 30]. The majority of cryptocurrencies are decentralized systems built on *blockchain* technology, a public tamper proof transaction ledger. With blockchain, anyone can verify the consistency of transactions without linking them to real-world identities. Satoshi Nakamoto is the founder of the first and most prevailing cryptocurrency from 2009, namely Bitcoin. In recent years, the number of other cryptocurrencies, often referred to as *altcoins*, have increased dramatically, and at the time writing there are over 2000 different cryptocurrencies [31]. Some popular altcoins are Ripple, Ethereum, XRP, and Litecoin. The altcoins describe themselves as improvements over Bitcoin, since Bitcoin face various complications which Subsection 2.2.1 details.

Traditional payment systems suffer from the inherent weakness of the trust based model. Completely non-reversible transactions are not possible since financial institutions cannot avoid mediating disputes. With the possibility of reversal, the need for trust spreads making merchants prompt customers for their confidential [32]. In contrast, cryptocurrency transactions are irreversible. Bitcoin defines an electronic payment as a chain of digital signatures, where each transfer are digitally signed with the previous transaction [32, 33]. Cryptocurrency systems are pseudonymous; the public sees all the transactions, but without being able to link them to real-world identities (Figure 2.4).

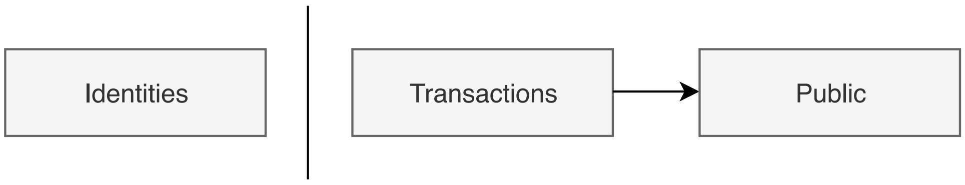


Figure 2.4: privacy model of Bitcoin. The transactions are public, as real-world identities are kept screened (source: [32]).

However, problems are originating from the privacy model of Bitcoin and the majority of altcoins. According to [3], the emergence of cryptocurrencies has raised significant concerns about potentially illegal activities, such as terrorism and money laundering, customer theft, and fraud. The expansion of cryptocurrencies may also threaten the traditional money issuance system, question the role of banks and other financial institutions in funds transfers, and present a risk for financial stability in general.

2.2.1 Blockchain

A blockchain [34] is an ever growing list of blocks [34]. Blocks in a blockchain consist of data and a *hash pointer*, a reference to and a cryptographic hash of the previous block, see Figure 2.5. Whereas a regular pointer makes it possible to retrieve a block's location in memory, a hash pointer also makes it possible to verify the integrity of the data. In other words, it is possible to check if the data within a block have changed after creation. In the context of cryptocurrency, blocks contain metadata and a series of transactions.

For nodes in cryptocurrency systems to solve the double-spending problem and agree on the succeeding block of pending transactions, they all must agree on a single block that comes next in line in the blockchain. The majority of

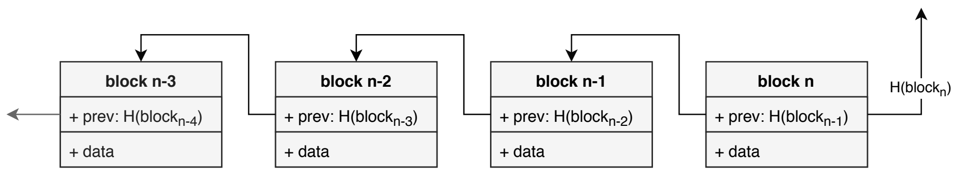


Figure 2.5: blockchain architecture. The blocks are structured like a list. A block in the chain contains a reference to and a hash of its previous block.

cryptocurrencies use the consensus protocol Proof-of-Work (POW) to elect the next block, which we define as follows: Miners collect broadcasted pending transactions into a block; they are fussy and exclusively pick the transactions with the highest fees [35, 36]. Then, the miners must solve a cryptographic puzzle. They start hashing the new block with the hash of the previous block until the digest is below a defined threshold. Each try is pseudo-stochastic, so it requires indefinite attempts, and miners can flip a bit in the block-field *nonce*, so they do not reuse the same digest. The first miner to solve the puzzle gets a minting reward, by broadcasting the block with its signature to the other miners who then add the new block into their blockchain. Statistically, systems that use POW retain their integrity as long as honest nodes possess more than 50% of the total hashing rate in the system.

After the immense interest in cryptocurrency, the number of miners has sky-rocketed. Which results in high electricity expenditure and longer verification time of transactions due to the slower propagation time of blocks. The Bitcoin miners electrical consumption alone can power five million U.S households, and the emission of CO_2 for producing the required electricity is 275 kilogram per transaction [37].

As the number of miners escalates, the longer propagation time of blocks, thus, the time-gap of where two or more block being proposed and integrated by miners at the same time expand [36]. When miners have a different vision of the blockchain, they have created a *branch*. Any branch is fixable though producing the longest branch as the POW protocol looks at the longest branch as the real main branch. The blocks that are mined but was cut off from the main branch are invalid and called *orphaned block*. Also, because of the possibility of orphaning, a rule of thumb in the verification of Bitcoin transactions is to wait until a block with your payment is confirmed (buried under other blocks) six times [38, 39] which is around one hour.

2.2.2 Exchanges

Cryptocurrency exchanges are centralized online platforms where traders can exchange cryptocurrency for another cryptocurrency or fiat[1]. They are market makers that lists coins over *bid-ask* spreads [40]. A bid-ask spread is the amount by which the asking price exceeds the bid price for an asset in the market. The bid-ask spread is essentially the difference between the highest price that a buyer is willing to pay for an asset and the lowest price that a seller is willing to accept [41]. Currently, these exchange lack regulation[2] which makes them not trustworthy and susceptible to price manipulation schemes and con artists [42, 43]. There are over 200 different cryptocurrency exchanges where some of the most appealing are Coinbase, Binance, Bittrex, and Poloniex [44, 45], where Binance alone has a monthly trading volume of more than $20 billion [46].

Cryptocurrency exchanges list various symbol pairs denoting a *base* and *quote*. The currency pairs compare the value of one currency to another - the base currency versus the quote. It indicates how much of the quote currency is needed to purchase one unit of the base currency [30]. For example, to trade Ethereum (ETH) for Bitcoin (BTC), the symbol pair would be "ETH/BTC".

Trades on cryptocurrency exchanges happens internally on exchanges; every coin is *tokenized*. Making trades to be off-blockchain [47], and in return, and there is no need for verification as trades happens instantly. Traders seemingly prefer to use multiple exchanges simultaneously, and transaction between exchanges are registered on the cryptocurrency's blockchain, see Figure 2.6. Exchanges are contradictory to the incentive of decentralized cryptocurrencies, as they are centralized. However, 99% of cryptocurrency trades still happen on exchanges [4].

Market data

Before the internet, trading took place over the phone, and now in the post-internet age, trading takes effect over an exchange's Application Programming Interface (API) [48] allowing any software to pull data and interact with the exchange. APIs are useful in terms of extracting data and do analytics with it and for traders who have algorithmic models that are fueled by live data and need to issue orders within milliseconds.

1. Money made by the government[2]
2. Application of law by the government

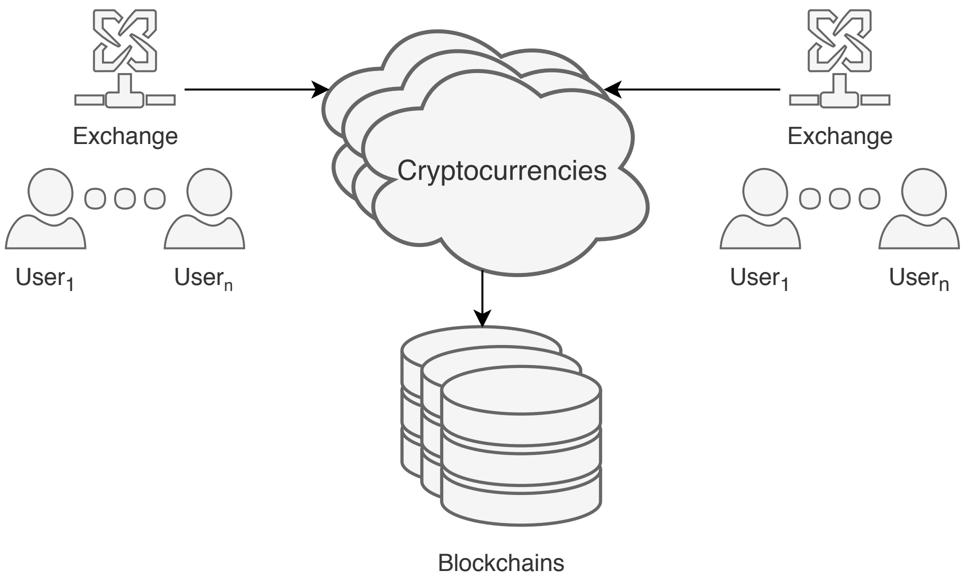


Figure 2.6: illustrates that exchanges incorporate numerous cryptocurrencies into their platform and that investors trade assets with exchanges' tokens.

OHLCV

Graphical illustrations of price movements for specific time intervals goes by the name *kline* or *candlestick* chart. Such graphs utilize a set of Open, High, Low, Close, Volumes (OHLCVs) points, describing the trading trends in a time window. Table 2.1 illustrates a candle in its crude form, and Figure 2.7 shows processed OHLCVs values making a candlestick chart. A kline's top and bottom wicks represent the highest and lowest trade price in its time interval, while the color portrays whether the closing price was higher or lower than the opening price [1]. Candles can define trading trends in any time interval, but exchange's API mostly allows a discrete selection of timeframes that commonly range from one minute to several days.

Timestamp	**Price**				**Trading volume**
	Open	**High**	**Low**	**Close**	
2019-06-01 23:59:59	2006.2	2061.3	1926.2	1984.1	216304.7

Table 2.1: shows the structure of an OHLCVs value. The volume presents the number of assets that were traded over a period. The price field denotes the highest and lowest price that was recorded, as well as the price from where the period started (open) to where it ended (close).

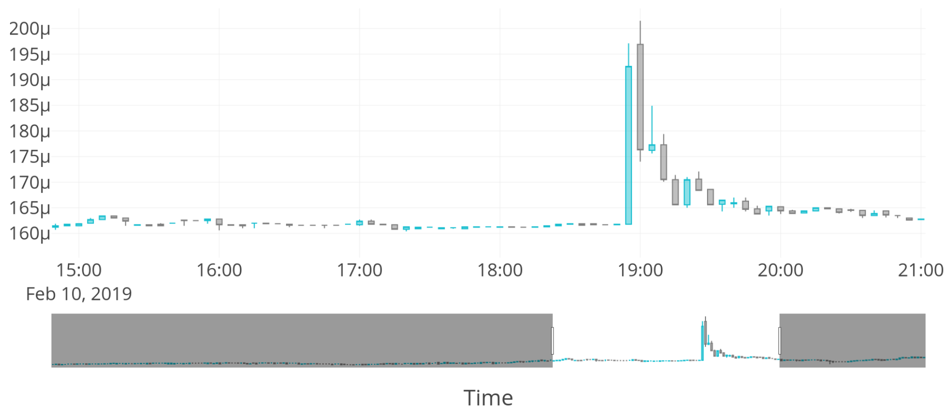


Figure 2.7: A real P&D organized by the group Crypto Pump Island on 2019-02-10 19:00. The target symbol was GXS/BTC on the exchange Binance.

Order Book

An order book, also called market depth, is an electronic list of buy and sell orders on an exchange for a specific symbol pair, see Table 2.2 [49]. Sell orders are in an *asks* list in a descending order while buy orders are arranged descendingly in a *bids* list [50, 51]. The order book is dynamic and constantly change throughout the day as traders issue new orders. There are many ways to interpret the information in an order book; for example, a massive imbalance of buy orders versus sell orders may indicate a price increase due to buying pressure [50].

Asks		**Bids**	
Price	**Volume**	**Price**	**Volume**
[0.028 ... 0.14]	[12.4 ... 3.1]	[0.027 ... 0.018]	[56.4 ... 1.45]

Table 2.2: shows the structure of an order book. The left side (askers) includes the quantity of coins that are buyable, while the right side (bidders) are bids that are waiting to be matched against buyable coins.

2.2.3 Pump-and-Dump

A P&D scheme is a type of fraud in which the offenders accumulate a commodity over a period. Then artificially inflate the price through means of spreading misinformation (pumping), before selling off what they bought to unsuspecting buyers at the higher price (dumping)[1]. After the emergence of cryptocurrency trading, P&D has become a popular legitimate price manipulation scheme among scammers, who leach assets from the misinformed.

Two researchers at the Imperial College London revealed that on average, at least two P&D schemes are carried out daily, producing roughly $7 million in daily trading volume [52]. Price increases of up to 950% have been witnessed, demonstrating the amount of potential profit [53].

On the modern stock market, P&D organizers focus on *penny* or *microcap* stocks, which are smaller companies that do not comply with the standards to being listed on the more comprehensive exchanges [5, 54, 55]. Microcap stock exchanges are not held to the same standard of regulation, which implies that there are usually not as much information about the companies that are listed, making them easier to manipulate. Misinformation about the stocks is usually spread through email spam, which has a net positive effect on the stock price [54]. It is illegal to run price manipulation schemes on regulated markets, and there are multiple cases where participants in a P&D have been prosecuted [1].

There is a slightly different approach for P&Ds on the cryptocurrency market. The pump is a coordinated, intentional, short-term increase in the demand of a symbol pair [5], organized by dedicated groups. These groups are often public channels in chat applications like Discord or Telegram and are joined by naive traders, who believe they will become wealthy in a short amount of time. There are designated web pages and forums that contains information and statistics regarding groups' P&D success, one of these pages are PumpOlymp[3]. The number of members in some of the prominent groups have peaked at around $200,000$ [56].

Pump Groups

In the cryptocurrency market, P&D organizers create groups in an encrypted chatting application such as Telegram and Discord. They advertise themselves through social media platforms and forums [6] as intriguing groups that ensure profit with little or no risk of losing assets. The group admins start to organize P&D when the group typically consists of over 1 000 optimistic traders. Only the admins are allowed to post messages in the group restricting regular members to see the messages posted by the admins; this functionality is enabled by the admins to avoid member interference [5].

Before a P&D, the group admins announce details regarding it a few days ahead. The information they provide is the exact time and date, the pairing coin, which is more or less always Bitcoin, and the exchange. With the information, the member can buy sufficient funds on the targeted exchange in advance. The

3. https://pumpolymp.com/

same day the P&D is scheduled, the admins purchase a commodity in the base coin over a period without raising the price. Then they send out countdowns and reminds the members of previous successfully P&Ds to motivate them to participate, and the rules during the upcoming P&D. According to [5], the standard rules are.

1. Make sure to buy fast.
2. *Shill*[4] the announced coin on social media to attract outsiders.
3. *HODL*[5] the coin for several minutes to give outsiders the possibility of joining.
4. Sell in pieces at profit, not in chunks.

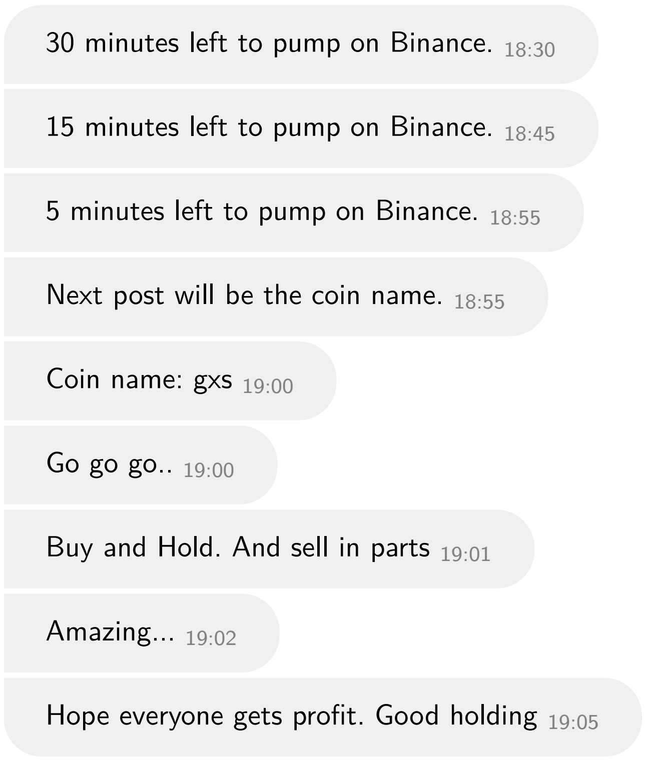
When the admins announce the coin, each member tries to be the first to buy the published coin to ensure profit before the inevitable inflation. If they are to slow, they might buy at the peak and are unable to make a profit. The pressure of being the first is high because the coin peak within seconds to max ten minutes [56]. After they have bought a significant amount of the coin, they shill, in an attempt to trick outsiders into buying it, allowing them to sell easier. The misinformation varies, but some common tactics include false news stories, non-existent projects, fake partnerships, or fake celebrity endorsements [56]. Simultaneously, the admins encourage the members to hold while they sell off what they bought earlier on that day, making them maximize their profit before the inevitable price dump. As soon as the first fall in price appears, the members start to panic-sell. If the price dips below the start price, the second wave of traders buys to bounce the price up to where it began allowing them to gain a small profit [5].

Minutes to hours later, when the coin recover its initial state. The admins publish results that showcase the members' impact and the potential profit. Figure 2.8 shows real messages from P&D organized by the pump group Crypto Pump Island[6], and the Figure 2.7 shows the impact.

Nevertheless, in the end, only the admins and a few members are profiting from a P&D while the majority are loosing. So why are there still people enthusiastic about partaking a P&D, given the risk of being ripped off by the admins? Because people believe that there are greater fools out there, who would buy the coins at an even higher price than their original purchase

4. Cryptocurrency jargon for "promote" or "advertise" coin.
5. Cryptocurrency jargon for Hold.
6. https://t.me/crypto_pump_island

price [5]. The greater fools theory is also what thrives many other price manipulation schemes [57].


30 minutes left to pump on Binance. 18:30

15 minutes left to pump on Binance. 18:45

5 minutes left to pump on Binance. 18:55

Next post will be the coin name. 18:55

Coin name: gxs 19:00

Go go go.. 19:00

Buy and Hold. And sell in parts 19:01

Amazing... 19:02

Hope everyone gets profit. Good holding 19:05

Figure 2.8: messages from the telegram group Crypto Pump Island on 10 February 2019.

Characteristics

Detection of P&D schemes requires insights in their operations to have the ability to identify patterns that occur during a P&D. Table 2.3, defined by two researchers at the University College London [5], summarizes some of the fundamental similarities and differences with respect to the target, tactic and timescale of traditional penny stock and cryptocurrency P&D schemes. It clearly shows that traditional and cryptocurrency P&D schemes target the same type of markets, but the tactic and timescale differ. The lack of trust among members in the pump groups can explain the short timescale of cryptocurrency P&Ds, as they all want their piece of the cake and sell as soon as they profit instead of holding. All the spreading of misinformation must happen in real-time because of the short time pressure.

Using these characteristics, the same two researchers [5] formulated criterion's that can be helpful when detecting P&D patterns in exchange data (Table 2.4).

	Traditional	**Cryptocurrency**
Target	Low market cap	Low market cap
	Low volume	Low volume
	Low price	Low price
	Lack of reliable information	Lack of reliable information
Tactic	Misinformation	Real-time
	Privately organized	Public or private group scams
Timescale	Medium (days to weeks)	Short (Seconds to minutes)

Table 2.3: Characteristics of traditional and cryptocurrency P&D schemes. (Source: [1]).

The indicators are split into *breakout* and *reinforcers*. The breakout indicators point out patterns that are present during the beginning of a P&D. And the reinforcers are external aspects to strengthening our confidence in an alleged P&D. The signs (+) and (−) are a confidential boost; the former denotes an increase while the latter denotes a decrease. The volume and price factors in the breakout indicators are discussed with an estimation window, referring to a collection of previous data points, of some user-specified length [5].

Breakout indicators	**Real-time indicators**	**Post-pump indicators**
Volume	Has the volume at the current data point been significantly higher than in the estimation window?	Was there a decline in volume after the event window where a pump was detected?
Price	Has the price at the current data point been significantly higher than in the estimation window?	Was there a decline in price after the event window where a pump was detected?

Reinforcers	**Temporal dimension**
Market cap	Is the market cap of the coin relatively low? (+)
Number of exchanges	Whether the coin is listed on multiple exchanges and the indicators only spike on one (+) Whether the coin is listed on multiple and the indicators spike on multiple exchanges (neutral) Whether the coin is not listed on multiple exchanges (+)
Symbol pair	Whether the coin is trading for BTC or some other cryptocurrency (+) Whether the coin is trading for USD or some other fiat currency (−)
Time	Whether the coin pump is on the hour(+)

Table 2.4: Indicators of P&D per temporal dimension and indicator type (Source: [1, 5]).

2.3 Deep Learning

Machine Learning (ML) is a fast-growing field in computer science. It refers to the ability to make a computer recognize specific patterns in data using various complex algorithmic models. In the broader field of Machine Learning (ML), recent years have witnessed a proliferation of deep neural networks, with fantastic results across various application domains. Deep learning is a subset of ML that achieves excellent performance and flexibility that surpasses conventional ML algorithms [58, 59].

A breakthrough in Machine Learning would be worth ten Microsoft.
- Bill Gates

As ML has become a buzzword and categorized as state of the art and "the solution" for every kind of problem. However, it is important to remember that ML is not always the optimal solution for every type of problem. There are certain cases where rule-based solutions perform better than ML, cases where we can directly predict values by using simple rules, computations, or predetermined steps that are easily programmable [9]. So when should we use ML? According to [9], we should use ML in following situations:

- When tasks cannot be adequately solved using deterministic rule-based solutions. A considerable number of factors like features, patterns, correlated features, etc., can influence the answer. When rules depend on too many factors, and many of these rules overlap or need to be tuned very finely, it quickly becomes complicated to define these rules accurately.

- When tasks do not scale, e.g., manual detection of spam mail, which will be a tedious process if there are millions of emails. ML solutions are effective at handling large-scale problems.

Algorithms in ML are commonly subdivided into two major paradigms, *unsupervised* and *supervised* learning. In supervised learning, the algorithms require data and prior knowledge of each sample, called *labels* or *ground truth*, while the algorithms in unsupervised learning only need data. There are two approaches under supervised learning, *regression*, and *classification*. Both share the same concept of using data to make a prediction. Classification problems refer to the ability to recognize a discrete set of classes, while a regression problem, estimates a value from input data.

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with various levels of abstraction [10]. Each layer's output is the connected layer's input [58]. These deep learning models learn like many other ML models, by iteratively minimizing a cost function to adjust its internal weights using a training dataset. When the weights are appropriately adjusted, it evaluates its performance by making classifications on a test data set. ML model facilitates the automatic classification of data, which when deployed, removes the need for a person to classify the data manually [58].

Considering the amount of P&D events in trading data, a P&D can be categorized as an anomaly as it exists significant more regular trading activity than P&Ds. The choice of a deep neural network architecture in anomaly detection methods primarily depends on the nature of input data. One distinguishes input data into sequential data (e.g., voice, text, music, time series, protein sequences) and non-sequential data (e.g., images) [59]. Cryptocurrency sources produce sequential data, more specifically *time series* data. Time series data are linearly ordered sequence of values of a variable at equally spaced time intervals [60]. Anomaly detection in multivariate time series data is a challenging task, but Recurrent Neural Network (RNN) and Long Short-term Memory (LSTM) networks are shown to perform well in detecting anomalies within time series [59].

2.3.1 Neural Networks

A traditional ML model do not have any perception of its previous predictions, and when working with non-sequential data, it is not needed. However, when having sequential data, a point only yields information within a specific context. For example, humans do not start thinking from scratch every second. When we read an essay, we understand each word based on our understanding of previous words. We do not throw everything away and start thinking from scratch again. Our thoughts have persistence [61]. The same concept can be applied to cryptocurrency markets; we do not know if the price of a coin has increased unless we know the previous values. Hence, if we attempted to detect P&D without having any perception of change in price, both humans and ML algorithms would have a hard time detecting them.

An Artificial Neural Network (ANN) (Figure 2.9) is a powerful learning model, and it is one of the most frequently used model [62] that achieve state-of-the-art results in a wide range of supervised learning tasks [63]. A vanilla ANN does not have a perception of previous samples. Thus, it can not catch trends in data. RNN is a type of network that has an internal state allowing it to set samples in a specific context. Yet, a standard RNN suffers from *vanishing* and *exploding* gradient problems [64]. LSTM networks is a type of RNN that solves said problems by using LSTM cells instead of standard RNN cells. The structure of LSTM networks is similar to Figure 2.9, and compose of three different layers, called *input layer*, *hidden layer*, and *output layer*.

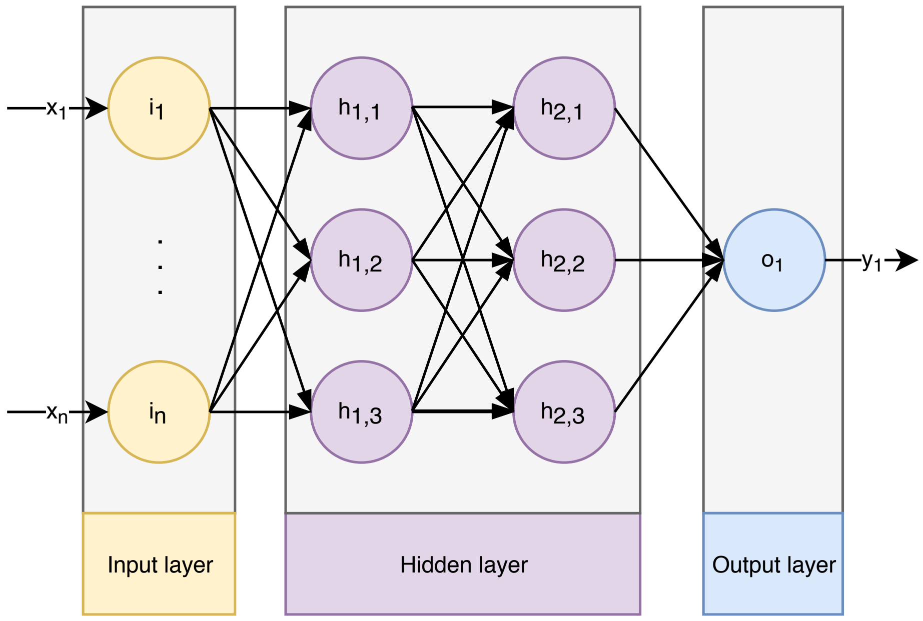


Figure 2.9: structures an artificial neural network. The passive cells in the input layer propagate their input data x to the layer of cells in the hidden layer. From there on, the cell's output in each layer is the succeeding cell's input in the next layer until the output layer returns a prediction y.

The cells in the input layer are mostly passive, meaning they do not make any calculations nor modifying the data, and the number of cells is equal to the number of features in the data. Their task is to duplicate their received data and propagate it to every cell in the first column in the hidden layer.

The hidden layer has no interaction with the outside of the network. Hence, the name hidden [65, 66]. This layer can comprise multiple layers with an arbitrary number of cells but with more cells, the complexity of the network increase. Each cell's algorithmic structure depends on the type of network we want to build(ANN, RNN), *perceptron* and *lstm* are two cells that are often used, where LSTM is good at detecting pattern in sequential data, while a perceptron is good at detecting pattern in non-sequential. All cells have at least two things in common; they all have an input and an output, and the output from one cell is the succeeding cells' input in the next layer.

The cells in the output layer take input from the last set of cells in the hidden layer. Their algorithmic structure is similar to the cells in the hidden layer, but their output is also the final prediction from the network.

One can build a more complex network than in Figure 2.9 by extending the number of hidden layers, where each hidden layer composes of even more cells.

There is no limit in terms of layers and cell, but with a deeper network, then the time it takes to propagate input data through the network increases. And layers with too many cells suffer from the *universal approximation* problem [67, 68]. So, determining the number of cells and layers actually boils down to multiple trial and error attempts.

2.4 Related Work

An article from International Conference on Advanced Computational Intelligence (ICACI) 2016, presented a model that detects P&D schemes on the stock market with 88% accuracy [69]. The article describes how badly P&D schemes are executed and organized. And from the patterns a P&D scheme leaves behind, the article proposed mathematical definitions based on level 2 order book data with a depth of 10 to generate a training set consisting of buying and sell orders. The researchers implemented a feedforward neural network and trained it with the generated dataset, and achieved an accuracy of precisely 88.28%.

Two students from Standford Unversity recreated the stock market model [69] and made it compatible with cryptocurrency exchanges in 2017. In their work, they used level 1 order book data to generate a training set to train a neural network and a Support Vector Machine (SVM). They labeled the dataset by identifying P&Ds by comparing a market's price movement to BTC. The final test results of their models had an accuracy of 78.13% with SVM and 82.5% with the neural network. Their research is interesting, they show that the order book alone is valuable, but they seem to ignore other relevant extractable data, also entirely ignore data preparation. As a result, we achieved significantly higher accuracy than them.

Kleinberg and Kamps from the University College London [1] defined specific patterns in P&Ds and how they differ from the stock market. Also, they proposed a novel anomaly detection approach based on a set of criteria for locating suspicious transaction patterns on cryptocurrency exchanges. The most balanced parameters for their algorithm resulted in about 1.6 P&Ds per market per day, for a total of 2150 P&Ds over 20 days of data. Moreover, 75% of the alleged P&Ds were found to have corresponding price dumps. They state in their conclusion; Ultimately, it is the hope that the information presented in this paper is useful for further research into the detection of this fraudulent scheme [1]. For us, this article proved to be indeed helpful as we use their anomaly detection algorithm to find P&Ds. Also, the P&D patterns they define is essential in terms of the features we need to possess.

Researchers from the Imperial College London wrote an article that analyzes features of pumped markets and market movement of coins before and after P&Ds [5]. They implemented a predictive Random Forest model that gives the likelihood of each possible currency being pumped prior to the actual pump event. With a probability curve of slightly over 0.9, the model exhibits high accuracy and is indicative of the probability of a coin being pumped. This article has actually received a lot of recognition and attention [52, 53, 70] on media platforms that cover cryptocurrencies. Their research in terms of how the organizer tends to build up P&Ds was invaluable for us when generating features. They also presented their model's feature importance, and the most prominent feature is a coin's capitalization.

3 Limelight's Design

This chapter presents Limelight's design. First, it briefly describes the system and all the components within it. Then, it goes further into details regarding every component in terms of their structure, function, and intent. Finally, it describes the two phases Limelight has.

3.1 Overview

We are modeling Limelight as a Machine Learning (ML) pipeline with some few extensions. Limelight is designed to extract raw data from numerous cryptocurrency sources and transform it into valuable features in order to classify Pump-and-Dumps (P&Ds). The term ML pipeline can be misleading as it implies a one-way flow of data when some elements in the pipeline are cyclical and iterative where every iteration intends to improve the accuracy of the model [71]. An illustration can be seen in Figure 3.1.

The first step in the pipeline pulls data over the internet from sources that has information regarding P&Ds in cryptocurrencies. The data is mostly incomplete by lacking trends or being unprocessed, making it potentially challenging to train a model and obtain good results. This process is tedious because sources tend to have different request rates, Application Programming Interfaces (APIs), and the data can have various formats like JavaScript Object Notation (JSON) or Extensible Markup Language (XML).

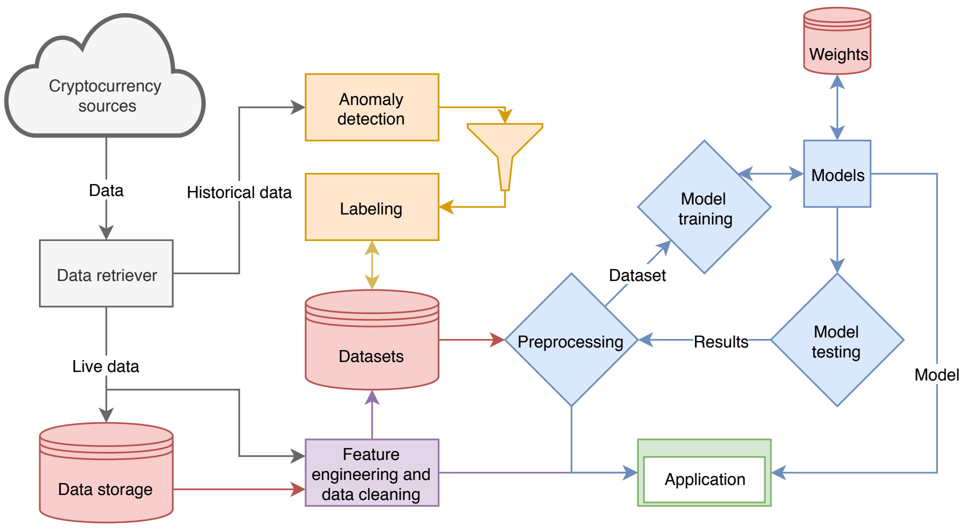


Figure 3.1: Limelight's architecture. The first processing stage in the pipeline is the data retriever, and it pushes data to the feature engineering and data cleansing stage to prepare the data for ML. Historical data that spans over the period where data is collected gets fetched and pushed through an anomaly detection algorithm that detects P&Ds. The labeled data gets first preprocessed, then either pushed to train a model or, if deployed, to straightly to a trained model that can make predictions.

The next step branches the retrieved data in two, live and historical data. Since we are trying to detect P&Ds in real-time we need to store live data continuously. When we have captured a compelling amount of P&Ds, we use an anomaly detection algorithm [1] to detect P&Ds in the gathered live data. This algorithm is not compliant with live data, so we need to pull aggregated historical data that span throughout the collected live data. As previously mentioned, anomaly detection algorithms tend to have a high number of false positive compared to true positive. Thus, we need to remove these false positives and keep the true positive manually.

The input data ultimately determine the performance of a ML deep learning model [58]. Training a model with the raw gathered live data is ineffective. Hence, we need to define a new convenient dataset containing features created by processing the collected live data; this is a highly critical process and will later determine the classification performance of the deep learning model. The gathered live data also need to undergo a cleansing process as a portion of it presumably are not relevant P&D information.

With filtered anomalies containing P&D, we create a labeled dataset and train our model. Obtaining good classification results depends, as mentioned, on the features, but also how we decide to preprocess the dataset. Typical preprocessing strategies include dimensionality reduction and normalization. Having a labeled preprocessed dataset, we can finally begin to train the deep learning model. This a cyclical and a long process, as it requires many trial and error attempts to find the optimal weights for the model. In each cycle, we store the model's weights because it is not always the case that each iteration will improve the classification performance of the model.

For applications to utilize Limelight, they need to select a model and let live data flow through the same processing stages as the dataset that was used to train the model.

3.2 Internal Components

3.2.1 Retrieving Data

Every problem that is solved using ML requires data. The more data, the better the results will be when training a model. As previously mentioned, cryptocurrency sources like exchanges produce time series data containing, e.g., price and volume of a coin. The data is continuously produced in a limited amount with proportion to time.

Since we want to detect P&Ds on exchanges in real-time, the nature of the data we want to make classifications on is live fine-grained data so that the model can detect them as early and accurately as possible. Aggregated historical data is too coarse-grained because exchanges generally only allow a discrete time interval selection of data where the smallest is typically one minute. The duration where they start to where they peak varies from a few seconds to a maximum of ten minutes [1, 52], and the ability to make accurate predictions with one-minute data is questionable.

Training a model in real-time by pulling data is impractical because it will not be labeled. In addition, sources can only produce a limited amount of data at a time which will create a bottleneck of data supply to the model. To cope with these problems, we have to pull and store current live data continuously; this is a time-consuming process for we have to wait until we have captured enough P&D events before we can start training. If anything fails, we may have to start all over again as we are missing out on trends, which results in noisy data.

From the reinforcers field in Table 2.4, we see that we have to fetch data from various sources. An exchange alone does not produce data regarding a coin's capitalization, nor a coin's price on a different exchange. Sources other than exchanges produce such metadata of coins, while exchanges only produce internal trading data.

Master Slave Approach

We shape our data retriever like a master/slave model. Figure 3.2 is an example that illustrates our data retriever with a master and its three data-pulling slaves. Each slave in our data retriever is assigned a source that can e.g. be an exchange. The communication between slaves and the master is as follows. The master broadcasts a pull signal to the slaves, and they pull the data from their assigned source. Then the master gathers the data from them and parses it, and augments all the data into a single sample. Each sample the master generate gets stored. This procedure takes effect in a fixed interval. By letting the master signalize the slaves to retrieve data simultaneously, we collect clean time-series data where each sample's time gap is circa equal.

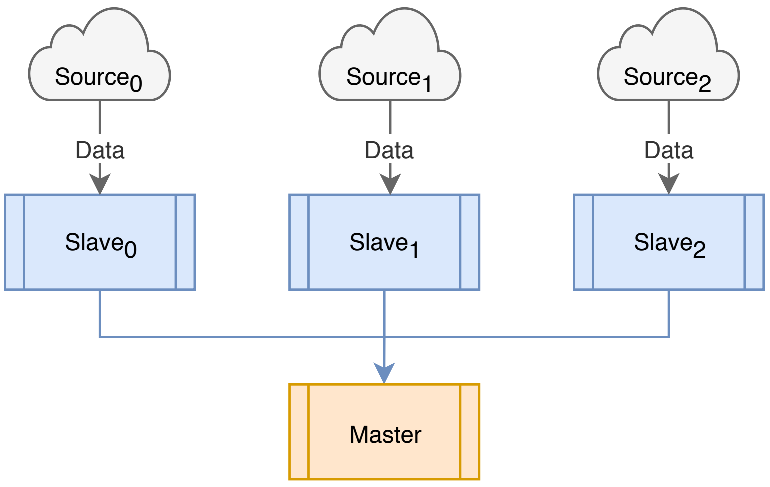


Figure 3.2: Limelight's data retriever. The slaves are assigned a source. The master, in a fixed interval, broadcasts a pull signal to the slaves, and the slaves fetch data from their designated source. Soon as the slaves complete their task, the master gathers the result from them.

Collecting Trading Data From Multiple Markets

Previous work in detection of P&Ds [1] estimated that 1.6 P&Ds is carried out daily per market, and this raises several problems. First, multiple exchanges have the same market, and we can not know which exchange they target unless we have prior knowledge from their P&D group. Second, gathering data from a single market is inadequate, the data retriever would with an estimate capture

48 P&D occurrences, if it gathered every P&D event on a single market for a whole month. Training a model with 48 P&D instances is inadequate.

To alleviate these problems, we collect data for all markets from a single exchange to make sure we obtain as many P&D events as possible. We assume the P&D pattern remain the same across markets. Otherwise, we may run into trouble when training our model by having too few samples with various patterns. Also, by training the model using data from all the markets generalizes the model, which makes the model compliant to new markets who are excluded during training.

Feature Description

From the P&D indicators described in Table 2.4, we define a set of features in Table 3.1. We believe that these features contain the necessary information for a model to detect P&Ds. Features like a coin's capitalization are available at CoinMarketCap, while trading data like order book and Open, High, Low, Close, Volume (OHLCV) values are available at exchanges. A specific feature that is challenging to attain is aggregated Open, High, Low, Close (OHLC) values from multiple exchanges, as this requires us to request multiple exchanges simultaneously and aggregate the data.

Feature	**Description**
OHLCV	Latest OHLCV values.
OHLCV multiple exchanges	Aggregated OHLCV values from multiple exchanges.
Order book	Level 1 (aggregated price and volume) order book with a depth of 5.
Order book imbalance	The imbalance between bids and asks orders and quantity.
Coin capitalization ratio	Coin capitalization ratio.
Volume traded	Base and quote volume traded for the last 24 hours.
number of trades	Number of completed trades for the last 24 hours.
bid and ask price	Best bid and ask price for the last 24 hours.
bid and ask volume	Best bid and ask quantity for the last 24 hours.
Average price	Average price for the last 24 hours.
symbol-pair exchange rate	The rate of how many exchanges that lists the symbol-pair.
Time	Unix timestamp.

Table 3.1: Features we believe are fundamental in the detection of Pump-and-Dumps.

3.2.2 Preparing Data

Data preparation is an important task. In practice, it has shown that data cleaning and preparation takes approximately 80% of the total data engineering effort, as it requires solid domain knowledge of the subject. Data preparation comprises those techniques concerned with analyzing raw data to yield quality data. Some of the techniques are; data collecting, data integration, data transformation, data cleaning, data reduction, and data discretization [72].

Because of this, we fetch data from multiple markets to create a generalized model; we must prepare the data and make it equally scaled. Markets are distinct, and they have different trading price and volume, which makes predictions with these raw numerical values nonviable. If a P&D occurs on a coin with a low cost, and the price is increasing at 300%. Despite the high increase and profit, the coin is still almost worthless compared to other expensive coins, and the numerical price increase do not necessarily need to be that high. On the other hand, if the price of an expensive coin increase with only 1%, the numerical value can still be high despite the low percentage increase. Since the model is generalized, it is weighing each market equally. So using raw data like the price is infeasible, as a low price change in an expensive market reflects a massive change in a cheap market. Thus, we must transform all these raw market specific values to values that are general across markets.

Data Cleansing

Data cleaning, also called data *cleansing* or *scrubbing*, deals with detecting and removing errors, inconsistencies, and unproductive features from data in order to improve the quality of data [73]. Fetched data from exchanges includes additional features not defined in Table 3.1. These features are removed as they add nothing of value. Instead, they increase the number of dimensions and makes the data more complex.

Markets with little or no activity may have intervals containing *zero-data*. For example, if no investors have bought or sold assets in a specific interval, the exchanges tend to set the trading values to zero. These zero-values create significant spikes in the trend which we must address; otherwise, they disrupt the data. Besides, having zero-data does not make any sense, if the price is recorded to be zero, it means that the coin is free, which it is not.

We substitute every value that is zero by linearly interpolating each of them. Linear interpolation involves estimating a new value by connecting two adjacent known parameters with a straight line [74]. These two known parameters are non-zero values that are adjacent on each side of the zero-value. Thus, we form

Equation 3.1 with the known parameters (x_1, y_1) (previous non-zero value) and (x_2, y_2) (next non-zero value), y is the new value for some zero-value in point x.

$$y = y_1 + (x - x_1)\frac{y_2 - y_1}{x_2 - x_1} \tag{3.1}$$

Feature Engineering

Feature engineering is the process of using domain knowledge of the data to create features that make ML algorithms work. Feature engineering is fundamental to the application of machine learning and is both challenging and expensive, but when done correctly, it can result in wonders [75].

Processing Order Book

As previously mentioned in subsubsection 2.2.3, P&D organizers invest in the market before the P&D without raising the price. We believe that the order book in said market oscillates before the pump, and especially during the pump. Therefore, we calculate an imbalance between sell and bid orders; this is a multidimensional problem considering an order book contains both a list of prices along with its volume, as we saw in Table 2.2. Equation 3.2 reduces this multidimensional problem to a single value. p and v is respectively the lists of prices and volumes, and the annotations a and b denotes ask and bid orders. If Equation 3.2 yields a value between $(0, 1)$, it emphasize the bidders, if it yields a value between $(1, \infty]$ then askers, and if it yields 1 the order book is balanced.

$$imbalance = \frac{\begin{bmatrix} p_1^a \\ \vdots \\ p_n^a \end{bmatrix} \cdot \begin{bmatrix} v_1^a \dots v_n^a \end{bmatrix}}{\begin{bmatrix} p_1^b \\ \vdots \\ p_n^b \end{bmatrix} \cdot \begin{bmatrix} v_1^b \dots v_n^b \end{bmatrix}} = \frac{\langle P_a, V_a \rangle}{\langle P_b, V_b \rangle} \tag{3.2}$$

Processing Trading Data

We process trading data such as price and volume by calculating the Percentage of Change (POC). By doing this, we transform these values in each market to the same scale, and eliminate the need for the model to adjust to raw values. For example, if the coin ETH increase by \$10 from a starting value of \$100, and the coin ADA increase by \$5 from a starting value of \$50, then both coins increase by 10%.

$$pct(x, v_\gamma) = \frac{x - v_\gamma}{x} \tag{3.3}$$

We define the function pct as in Equation 3.3 to calculate the POC. It calculates the percentage change in point x concerning a previous value v with a time lag γ. We consider x and v to be a single value like the price or volume, while γ indicates moving backward in time from point x. We must be vigilant when using this technique, if the data contains values that are zero, we might perform a zero-division that can clutter with the data unpredictably. However, since we scrubbed the data first and removed zero-values, this should not be a concern.

Processing Time

According to [5], the time is essential when classifying P&Ds. They are typically executed on the hour (6:00, 7:00, etc.) because organizers usually does not choose a random time. Data retrieved from sources are getting timestamped, and we can take advantage of these timestamps to check whether data was generated at the hour or not. The function x_δ transforms a Unix timestamp[1] to a value in the interval $[0, 1)$. The closer x_δ is to the margins, the closer the time is to the hour, but since 0 and 1 symbolizes the same, we have to process it further before we can use it as a valuable feature.

$$x_\delta(x_{\textbf{unix}}) = \frac{x_{\textbf{unix}} \mod 3600}{3600} \tag{3.4}$$

1. Unix timestamp is a way to track time as a running total of seconds since the Unix Epoch, January 1st, 1970.

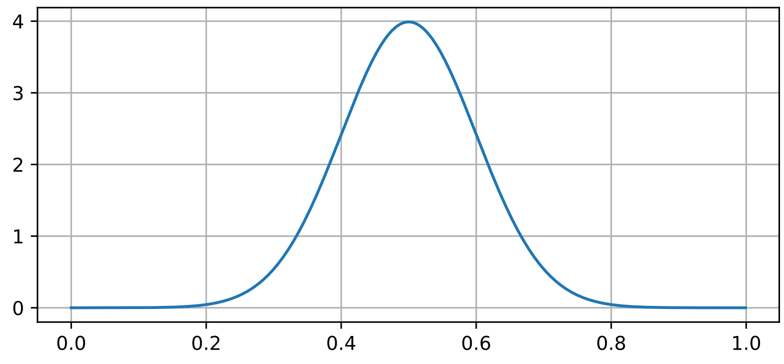


Figure 3.3: Gaussian distribution t from Equation 3.5 with the parameters $\sigma = 0.1$ and $\mu = 0.5$. The curve describe how close a given time-parameter x is on the hour (6:00, 7:00, etc.).

$$t(x_\delta, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x_\delta - \mu}{2\sigma^2}}, \quad \text{where} \begin{cases} \mu = 0.5 \\ \sigma = 0.1 \end{cases} \tag{3.5}$$

We define a Gaussian distribution function t (Equation 3.5) with the parameters $\mu = 0.5$ and $\sigma = 0.1$ which creates the graph illustrated in Figure 3.3. The graph shows when given t a x_δ close to 0 or 1 (6:00, 7:00, etc.), t returns a value close to 0, but when given 0.5 (6:30, 7:30, etc.) it returns a value close to 4. t demonstrate that we can separate data by how close it was generated to the hour. Also, we can always tweak σ to adjust the width of the curve Adjusting the curve allows us to be more strict, the wider the curve, the more strict.

3.2.3 Collecting Pump-and-Dumps

Manually collecting P&D events from chat applications to label our collected data. seems infeasible in the long run. Although Livshits and Xu [5] hand-picked 220 pump-events from July to November in 2018 from 358 different Telegram groups to train a Random Forest model, they still missed a lot of other executed P&D schemes as there are numerous chatting applications and private organizers [76]. Also, searching for P&D events is a time-consuming process, and incorrect labeling occurs when we lack membership to all of P&D groups, which results in suboptimal prediction capacity [77]. We also have to be aware of whether a P&D was successfully executed or not; labeling failed attempts as positive only contributes to label noise. Instead of manually collecting P&Ds

from groups in chatting applications, we believe that it is possible to hand-pick P&Ds by *reasoning abductively* with the help of an anomaly detection algorithm to pinpoint suspicious time intervals in historical data.

The anomaly detection algorithm identifies local *contextual anomalies* based on fixed recent history called a *sliding time window*. A sliding window is a period that stretches back in time (lag factor) from the present containing events at specified intervals. The event intervals can overlap with each other, or they can be disjunct. As events exceed the lag factor, they fall out of the sliding window, and they are no longer matching against the rules applied to the sliding window [78]. With a sliding window, we can compare values in a given period, contrary to using single values, which alone does not yield much information in sequence data.

The anomaly detection algorithm is proposed by Kleinberg and Kamps [1], which is inspired by previous research in Denial-of-Service (DOS) attacks [79]. It is a threshold based technique to find a suspicious increase in price and volume of a coin. If the price and volume in a specific interval are higher than some threshold, then the interval is flagged as anomalous and warrants further investigation.

Price Anomaly

We compute the price anomaly threshold by the simple moving average listed in Equation 3.6. μ_γ^p of OHLCV values denoted x with a lag factor γ multiplied with a given percentage increase ϵ_p. We consider x and γ as OHLCV objects, and $x-\gamma$ indicates moving backwards in the sliding time window by a factor of γ [1]. If the highest registered price in x's period is greater than the computed threshold, we flag the period as anomalous.

$$\mu_\gamma^p(x) = \frac{\sum_{i=x-\gamma}^{x} x_{close}}{\gamma}$$

$$price\_anomaly(x) = \begin{cases} True & \text{if } x_{high} > \epsilon_p \cdot \mu_\gamma^p(x) \\ False & \text{otherwise} \end{cases} \tag{3.6}$$

Volume Anomaly

Calculating the volume anomaly threshold is almost identical to calculating the price anomaly. We are only substituting $x_{closing}$ and x_{high} with x_{volume}. The resulting formula is listed in Equation 3.7.

$$\mu_{\gamma}^{v}(x) = \frac{\sum_{i=x-\gamma}^{x} x_{volume}}{\gamma}$$

$$volume\_anomaly(x) = \begin{cases} True & \text{if } x_{volume} > \epsilon_v \cdot \mu_{\gamma}^{v}(x) \\ False & \text{otherwise} \end{cases} \tag{3.7}$$

Filtering Anomalies

Anomaly detection algorithms have a high false alarm rate [80], which makes them difficult to use. And since we want to use the anomalies we collected to train a model with, it is imperative to remove false P&Ds. Otherwise, training a model with false P&Ds will make the model perform poorer, and result in even higher occurrences of false positives. Therefore, we manually remove all the false-positive anomalies, but removing them requires prior knowledge of P&Ds, which we do not have.

The anomaly detection algorithm checks whether P&Ds occurs in a specific interval. We visualize one-minute klines that span over intervals that are flagged anomalous. By plotting these values we can compare them to real P&Ds, like the ones seen in Figure 2.7. A market's capitalization, if the pairing coin is BTC, is also helpful when filtering P&D. E.g. if the anomaly detection algorithm flagged a suspicious interval in the ETH-BTC market, and it visually looks like a P&D, then it still would with a high likelihood not be a P&D as ETH is the coin with the second highest capitalization [81]. As this would break the pattern where the organizers target coin with low capitalization. Further helpful characteristics regarding P&Ds is described in Table 2.3 and Table 2.4.

Labeling Pump-And-Dumps

With the new generated features from collected data and filtered anomalies containing P&Ds, we can label the new features and define a dataset. But first, the filtered anomalies are still intervals where a P&D occurred, and not precisely when it occurred. So, we need to precisely define where every P&D started and ended, before labeling.

Because the purpose of P&Ds is to raise the price of an asset, the point where the price change is highest in an interval is where the P&D peaked. If we have the peak of a P&D, we can search from the peak and down the descending slopes on each side of it. Left side will be the pump and the tight side will be the dump. From each side, we search until the change in price is equal or smaller than 0. Finally, when have defined the start and end of every P&D, we label all the new features positive where there are P&Ds, and negative otherwise.

3.2.4 Deep Learning

This section describes the model we use to detect P&Ds, and how we process data, trains the model, and which metrics we should use to evaluate the model. These steps are an iterative process, where each iteration tend to improve the model.

Preprocessing

Before training a model with the generated dataset, we transform the dataset by *normalizing* the features as they have various scales. Then, a common technique is to *normalize* the input data. Normalization creates new values from the dataset, but still maintain the general distribution and ratios in the source data while keeping values within a scale applied across all used features [82]. Also, normalization of data occasionally improves the performance of the model by accelerating convergence speed of its weights during training [83, 84].

There are several ways to normalize data, and the normalization technique used may have an impact on the performance of the model. Since P&Ds are anomalies, some features like percentage change in price and volume will make them look like outliers in the data. A traditional method called *min-max* normalization is repeatedly used in detection of outliers [85, 86]. This technique provides a linear transformation of the data [87], allowing us to keep the distance ratio, and it scales every value into the range $[0, 1]$.

$$x'_{ij} = \frac{x_{ij} - \min(x_j)}{\max(x_j) - \min(x_j)} \tag{3.8}$$

Equation 3.8 is the min-max normalization formula for a coefficient x_{ij} in a dataset x. x_j represents a column in a dataset with samples as row vectors.

Model

The model type we use to detect P&D is a Long Short-term Memory (LSTM) network, illustrated in Figure 3.4. This network contains LSTM cells in the hidden layer as we want the cells to have an internal state to remember the trend in data. We modeled the detection of P&Ds as a *binary classification* problem, it is either positive (P&D) or negative (not P&D). Thus, in the output layer we only require a single perceptron to make the final classification.

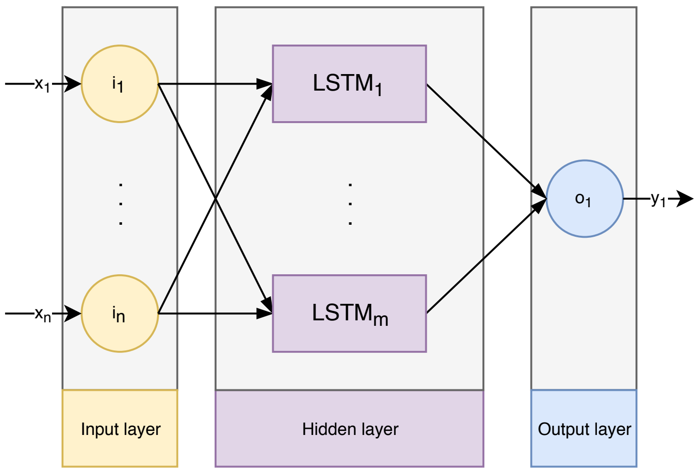


Figure 3.4: structure of our model. The passive cells in the input layer propagate their input data x to the first hidden layer that compose of LSTM cells. Then the cell's output in the hidden layers is the cell's input in the output layer. The last cell yields a prediction y.

Training

Before we start to train our model, we have to split the dataset into two sets, a training set, and a test set. The training set will we use to train the model, and the test will we use to evaluate the model. There is also a validation set that is used to fine-tune the hyperparameters during the training, but since there are so few P&D samples, we choose to avoid using a validation set and use those P&D samples we have to train our model and evaluate it.

As previously mentioned, P&Ds are anomalies, and that results in a significant class imbalance between positive and negative samples, which, when trained will make the model to overfit to the negative class and more or less ignore the positive class, it entirely depends on the distribution of them. Also, the academia is split concerning the definition, implication and possible solutions to this problem [88] making it currently ambiguous how to properly deal with it.

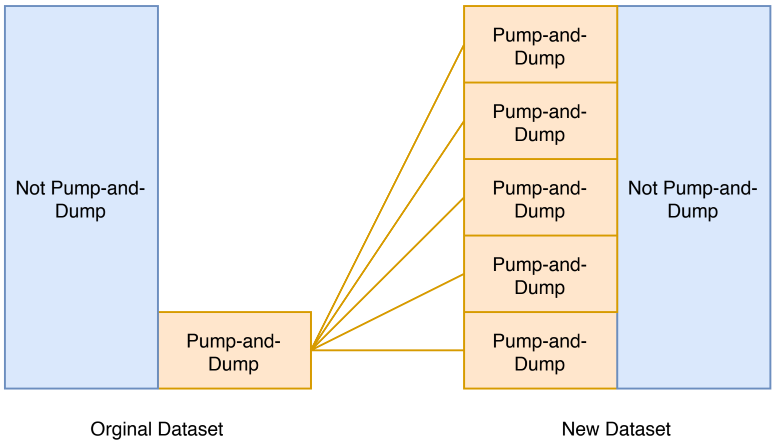


Figure 3.5: Oversampling (source[89]). Involves duplicating a minority class (P&D) until the size is equal to the majority class (not P&D).

We believe that we can solve our imbalance problem by using a technique called *oversampling*, which Figure 3.5 illustrates. Oversampling gathers all the samples from the minority class and duplicates them until the amount is approximately equal to the majority class. There is also a technique called *undersampling*, that select random samples into a subset from the majority class until the size is equal to the number of samples in the minority class, but by using the oversampling technique, we can train our model with the whole dataset. With a balanced dataset, we can finally train our model.

Depending on the size of the dataset and number of epochs, training a model takes a really long time. Thus, after each training, we store the model and all its weights and other internal parameters.

3.3 Phases

Till now, in this chapter, we have described each component in Limelight. As we mentioned in Chapter 2, when we have trained a model, we can feed the model with real-time data and it will classify whether the given data is a P&D

or not. This allows us to dismiss numerous components which are not needed anymore. They only need to be present during the training phase. So we can divide Limelight into two phases, a training phase, and a deployment phase. In the training phase, Limelight depends on all the components except the application box in Figure 3.1. In the deployment phase, there are only a few components needed.

3.3.1 Deployment

Figure 3.6 illustrates the components that are needed in this phase. The first component, the data retriever, function precisely like described above, it fetches data and pushes it further down the pipeline. The next two stages need to retain their internal parameters from the training phase. Otherwise, the model will classify input data that is processed differently from what it was trained with.

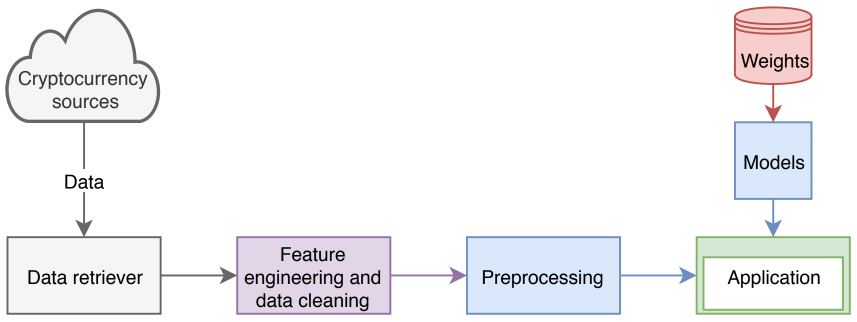


Figure 3.6: deployment pipeline. The data retriever fetch data and pushes it further to create new features. The new features get normalized and given to the application. With data and a model, the application can make a prediction.

In order for applications to use Limelight, they need to select a model that was generated during the training phase. With both a trained model and real-time data, they can potentially predict whether the given input data is a P&D or not.

4 Implementation of Limelight

This chapter describes the implementation of Limelight. First we will go through the programming language and modules we use. Then we will detail the implementation of each component.

We implement Limelight in the programming language Python, which is a powerful high-level scripting language that is excellent for building prototypes and conducting experiments. It has an extensive standard library, in addition to an active community which provides specialized software that implements various protocols, Application Programming Interfaces (APIs), etc. It has roots in the following programming paradigms, procedural, object-oriented, and functional. In recent year, it has climbed to the very top of the most popular programming language according to PYPL[1] [90].

Python has an interpreter, a program that executes code. Each python interpreter has a Global Interpreter Lock (GIL), which is a mutex that protects access to Python objects, preventing multiple threads from executing Python bytecodes at once. This lock is necessary mainly because of CPython's memory management is not thread-safe [91]. And because of the GIL, using a single interpreter to run code in parallel is impossible. To run parallel code in python,

1. Popularity of Programming Languge

we must initate multiple interpreters.

The modules we use are all available from PyPI[2], a python software repository, and they are installable by using PyPI's package installer `pip`. The following modules are used in Limelight:

- **ccxt** - package containing a uniform API that implements numerous exchanges.
- **keras** - high-level deep learning network framework.
- **numpy** - fundamental scientific computing module.
- **pandas** - library containing data structures and data analysis tools.
- **matpotlib** - plotting library.
- **time-series** - module for working with time-series data.
- **scikit-learn** - tools for data mining and analysis.
- **python-binance** - Python implementation of the exchange Binance.
- **CoinMarketCapAPI** - Python implementation of the cryptocurrency tracking site CoinMarketCap.

4.1 Data Retriever

We implement Limelight's data retriever as a master/slave architecture like described in Subsection 2.1.1. The master is the main process, and the slaves are threads. Hence, the master and the slaves are running inside a single interpreter. The slaves are assigned a source, and they continuously pull data from the markets as long as they do not exceed their designated source request rate. Then, in a fixed interval, the master collects data from the slaves. All the slaves are distinct objects, but they implement the same *interface*, allowing the master to exploit the concept of *polymorphism* when retrieving data from the slaves.

2. Python Package Index

Code Listing 4.1: Data retriever's slave interface

```
class Slave(Thread):
    def __init__(self, markets):
    def header(self):
    def start(self):
    def stop(self):
    def row(self, market):
```

code listing 4.1 is the interface each slave implements. The initializer method `__init__` initialize the internal state of each slave, and the argument `markets` is a list of markets the slaves need to fetch data from. `start` and `stop` makes the slave start and stop retrieving data. `start` method spawns a daemon that continuously retrieves data that gets cached in memory. `header` returns a list of feature names in metadata that gets extracted from the markets, while `row`, parse and return the latest cached data from a given market. As long as the daemon run and fetch data, the cache will always contain the latest data.

The execution flow of the data retriever begins with the master fetching coins paired with BTC from Binance. Then, the master initializes slaves with those markets. After the initialization, the master fetches each slaves list of headers to define the structure of the data, and have prior knowledge of data retrieved from its slaves. All the headers are augmented into a single header and stored as the first row in a file that identifies a market, creating a data frame like Table 4.1. The headers are defined as c and the data/feature as f. Then, the master and slaves go into an iterative phase where each iteration, the master retrieve the freshest data from its slaves' cache. The data then gets parsed such that each feature f gets appended to their corresponding file and column c.

	c_1	c_2	$\dots$	c_n
1	$f_{1,1}$	$f_{1,2}$	$\dots$	$f_{1,n}$
2	$f_{2,1}$	$f_{2,2}$	$\dots$	$f_{2,n}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
m	$f_{m,1}$	$f_{m,2}$	$\dots$	$f_{m,n}$

Table 4.1: A dataframe can be thought of as a dictionary-like container. Where the columns c, are names and represents keys, while the features f are values that are mapped to a column.

Naming and storing features in a dataframe is user-friendly in terms easy access to features. It is practical when processing the data by indexing features by name, instead of indexing the features numerically, which is a hassle as we would need to remember what each index means.

4.1.1 Sources

We implement three slaves who are assigned different sources. The first slave is pulling data from the exchange Binance by using the module python-binance; the second slave is pulling data from multiple exchanges with the help of the module Ccxt. Finally, the third slave pulls data from the cryptocurrency tracking site CoinMarketCap by using a module we create.

Binance

The exchange Binance provides various type of services through their API. We only need the newest trading data regarding the initialized markets. We use their WebSocket API that takes form as a *pubsub* client where Binance pushes data to us. The first function that we subscribe to is the *ticker* service, a 24-hour rolling window containing statistics for a symbol that gets pushed every second [92]. code listing 4.2 shows a ticker JavaScript Object Notation (JSON) response, and this response is retrieved for each market we subscribe to. In the ticker response, there are Open, High, Low, Close, Volume (OHLCV) values for the past 24-hour and other additional features that may prove themselves valuable in the detection of Pump-and-Dumps (P&Ds).

Code Listing 4.2: Ticker response from Binance.

```
{
  "e": "24hrTicker",  // Event type
  "E": 123456789,     // Event time
  "s": "BNBBTC",      // Symbol
  "p": "0.0015",      // Price change
  "P": "250.00",      // Price change percent
  "w": "0.0018",      // Weighted average price
  "x": "0.0009",      // First trade(F)-1 price
  "c": "0.0025",      // Last price
  "Q": "10",          // Last quantity
  "b": "0.0024",      // Best bid price
  "B": "10",          // Best bid quantity
  "a": "0.0026",      // Best ask price
  "A": "100",         // Best ask quantity
  "o": "0.0010",      // Open price
  "h": "0.0025",      // High price
  "l": "0.0010",      // Low price
  "v": "10000",       // Base asset volume
  "q": "18",          // Quote asset volume
  "O": 0,             // Statistics open time
  "C": 86400000,      // Statistics close time
  "F": 0,             // First trade ID
  "L": 18150,         // Last trade Id
  "n": 18151          // Total number of trades
}
```

Furthermore, we also want the order book from Binance, who has proved to be valuable by two researchers at Standford as they were able to detect P&D with an accuracy of around 80%. So, in addition to subscribing on the ticker symbol, we also subscribe to Binance's *depth* function. This function will return a response equal to code listing 4.3.

Code Listing 4.3: Depth response from Binance.

```
{
  "lastUpdateId": 160,  // Last update ID
  "bids": [             // Bids
    [
      "0.0024",         // Price
      "10"              // Quantity
    ]
  ],
  "asks": [             // Asks
    [
      "0.0026",         // Price
      "100"             // Quantity
    ]
  ]
}
```

Ccxt

The second slave uses the ccxt module, which provides a uniform API access to numerous exchanges. We use this module to fetch Open, High, Low, Close (OHLC) values across exchanges. There is a complication by using this module because it provides uniform access to over 100 exchanges, which becomes troublesome when the slave is initiated with the markets it has to fetch data from, and we do not have any prior knowledge of exchanges' markets, yet.

To abbreviate with this problem, we first create a reverse map, that maps markets to a list of exchanges. This was done by initiating all the exchanges from the ccxt module and request all the exchanges' markets and map every market to a list of exchanges. Furthermore, we defined a Publish/Subscribe (pub/sub) system illustrated in Figure 4.1, that enables synchronization of

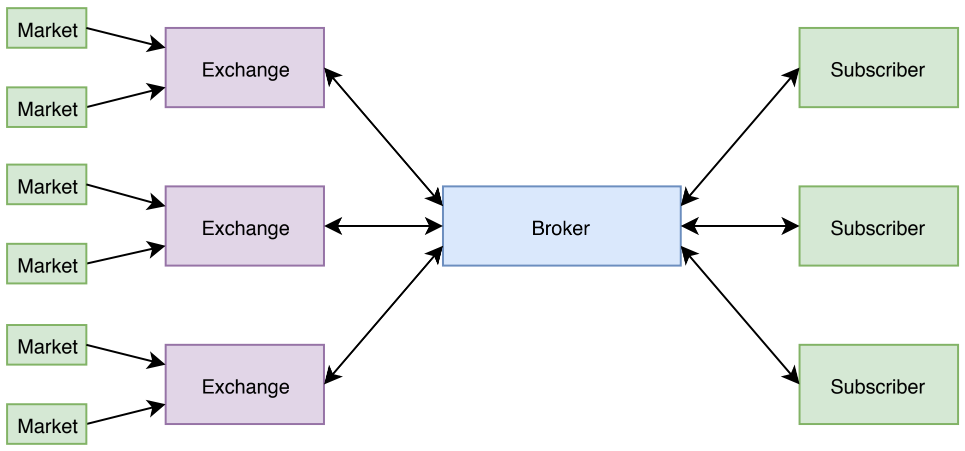


Figure 4.1: exchanges illustrate publishers; they extract market data that gets propagated to the broker. The subscribers can express interest in specific markets to the broker. And the broker forwards data from the publishers to the subscribers.

request issued to exchanges. From the Figure 4.1, the exchanges (threads) illustrate publishers that fetch market data and propagates it to the broker. Subscribers (threads) can issue interest in a specific market to the broker. The broker synchronizes with exchanges and makes sure them issues request simultaneously. When all the exchanges have propagated their response to the broker, it aggregates all the data from each coin. And finally, the broker pushes the data to the subscribers by using a callback function which was given when the subscribers first issued their interest in a market. Subscribers just wait for the master to request data from their cache.

The method we use to extract market data from ccxt's uniform API is the ticker function, which contains similar information given by Binance, but we only use the OHLC values. With the reverse map we created, we also estimate the ratio of how many exchanges that lists a specific market as this feature was explicitly described in Table 2.3 as a P&D characteristics.

CoinMarketCap

The final slave retrieves data from CoinMarketCap, which is a popular cryptocurrency tracking site and contains statistics regarding a coin's capitalization, circulation, etc. We must use CoinMarketCap's private API as they recently deprecated their public API. Using their private API requires an API key for user identification. With the API key, CoinMarketCap are able to track users request rate, which is troublesome as their request limit to a free subscription is low, only 333 request daily is allowed to issue. They have defined a severe complex request system. Where a user has a load of credits, and the amount of credit one has, depends on the type of subscription one has. A user also has a daily, weekly, and monthly request rate system, which also depends on the type of subscription on have.

There are some Python modules in existence that allows one to extract data from CoinMarketCap, but they completely ignore the request system, which when used will be problematic as we continuously will exceed the request limit and credit system, which ultimately results in a permanent ban from using their API.

Thus, we create a module that supports throttling of requests, which adjusts to both the credit system and the number of requests one can issue. And we cache every request in an SQLite database because CoinMarketCap mostly provides data that rarely or slightly change, and the cache functionality comes with an exceed factor.

These are the four endpoints at CoinMarketCap:

- Cryptocurrency
- Exchanges
- Globals
- Tools

We use the modules requests, requests_cache, and ratelimit. The requests module for making requests to CoinMarketCap's API. Requests_cache is a monkey patch to the request module, and caches every request made by it. The ratelimit allows us to define function decorators that restricts the number of function calls within an interval.

This slave fetches data from the endpoints, cryptocurrency and global. In the global endpoint, we use the global-metric function that contains the total market capitalization of all coins. And in the cryptocurrency endpoints, we fetch cryptocurrencies' latest market data that contains fields like capitalization, which says how a coin is worth. CoinMarketCap does provide market data, but since we cache every request and the request limit is limited to 333 a day, we can not use it as it results in us using stale trading data.

4.2 Data Preparing

After gathering enough data, we start to prepare it for Machine Learning (ML). Each market has a file containing data from the sources we recently described. And since we prepare the data in the files equally, we can speed up the operations by processing the files in parallel. Each file we have to process can be small in general, but combining them all, they become quite large, and it is infeasible to load them all into memory. Hence, we only process a fixed number of files in parallel.

We parallelize every operation by creating, yet again, a master/slave structure where the master process spawns the same number of child processes (slaves) as there are markets. The slaves are initialized with a market, a semaphore to control the number of processes to run inside a critical section, and a queue to signalize back to the master when it's done. Inside the critical section, the child loads the dataset, execute the desired operation, and stores the result. After the slave leaves the critical section, it signalizes back to the master. The slaves use the module Pandas to load the datasets into a dataframe. With a data frame, a slave can index features by their column name, which makes it convenient when we do cleansing and feature engineering.

The first operations involve cleansing the data. We cleanse by removing corrupted files. We also remove the features that are not needed, e.g, in code listing 4.2, the constant categorical field `symbol` are not needed when classifying P&D. The last cleansing process we do is to interpolate the data. Values involving price and volume gets linearly interpolated as we describe in Chapter 3.

The second series of operation we do is to create new features. We calculate the Percentage of Change (POC) like described in Chapter 3 on all the values we interpolated. Then we create new features from using the timestamp that we added along with the data when it was stored. Finally, we create a new feature that describes the imbalance of the order book.

4.3 Collecting pump-and-dumps

We implement the anomaly detection algorithm to detect anomalous intervals contains P&D, this algorithm was described in Chapter 3. The algorithm marks an interval anomalous if there is a significant increase in price and volume in that period compared to the previous periods. We can not use the anomaly detection algorithm with the real-time data that was collected, as this data is not compliant with the algorithm. Instead, we retrieve historical OHLC data from Binance that span over the period we gathered data.

This algorithm uses sliding a window containing a fixed number of OHLCV values. We created a sliding window module which wraps around a python list. The list has fixed size, and when adding new elements to the list, the last element will be removed, just like a fixed size First In, First Out (FIFO) structure.

code listing 4.4 illustrates a response from Binance that contains historical OHLC values. With these values and a sliding window, we start to continuously add OHLCV values that span over the period we collected data. And we iteratively calculate whether the newest added OHLC is an anomaly compared to the other OHLCV values in the window. The fields that we use from code listing 4.4 are `close` and `high` to calculate whether the price surpasses a specified threshold, while we use the field `volume` when calculating whether volume exceeds a specified volume threshold.

Code Listing 4.4: Historical kline response from Binance.

```
[
  [
    1499040000000,       // Open time
    "0.01634790",        // Open
    "0.80000000",        // High
    "0.01575800",        // Low
    "0.01577100",        // Close
    "148976.11427815",   // Volume
    1499644799999,       // Close time
    "2434.19055334",     // Quote asset volume
    308,                 // Number of trades
    "1756.87402397",     // Taker base volume
    "28.46694368",       // Taker quote volume
  ]
]
```

As we have previously mentioned, anomaly detection algorithms tend to have a high occurrence of false positive compared to true positive. So we remove the false positives anomalies by first plotting finer-grained OHLCs values over the periods that were flagged anomalous in each market. Then, removing the anomalies that we believe is not a P&D, while retaining the anomalies that we believe is a P&D. We plot the OHLC values by using the matplotlib module.

After filtering P&Ds, we label the features that we have created. We use the same master/slave method, which we used during the feature engineering stage. Where each process is initiated with a semaphore and queue, but also the P&D anomalies. To label the dataset, slaves loads its assigned dataset and identify the interval where the P&D occurred, and find the points where the most significant change in price percent happened. From the highest point, we label the points as positive from where the change in percent was positive until it becomes negative.

4.4 Deep Learning

We normalized each market using the min-max method described in Chapter 3. We have to process the data over two iterations, in the first iteration, we iterate over all the markets to find the largest and smallest value in the features. Then, in the second iteration, we normalize the features.

To create a balanced dataset, we undersample the data. So, we first have to collect all the positive samples from the datasets. Then we select random negative sequences from them until the number of negative and positive samples are approximately equal.

We use the deep learning library Keras for building the model, and the first layer contains Long Short-term Memory (LSTM) cells and the output layer contains a single perceptron. Training a LSTMs network requires us to reshape our 2D input data, into a 3D shaped matrix. The 3D shape represents matrices inside a matrix. A sample in a 3D space is a matrix containing a batch of vectors within a time lag. Then the third dimensions is a batch of these new samples. After shaping the data, we can finally train our model.

5 Evaluation

This chapter evaluates Limelight. It first describes how the experiment was executed. Then, we present the results. Finally, we discuss the results and propose potential improvements.

5.1 Experimental Setup

We started the experiment by retrieving real-time data from 138 different markets throughout 33 days, where every markets' pairing coin was Bitcoin. The interval between data points was 5 seconds, which resulted in approximately 570 000 samples per market. Over these days, we collected in total 47 GB of data. The sources that we used to fetch data from were those we presented in the Chapter 4, namely Binance, CoinMarketCap, and aggregated data from Ccxt.

Table 5.1 contains details regarding every feature that were fetched. The operation column describes how we processed features. We cleaned the data by removing features that we believed was unproductive and those were tagged as *Removed*. The features tagged Percentage of Change (POC), was calculated by interpolating the respective values between samples. We chose 10 minutes for the time lag, due to the period where a Pump-and-Dump (P&D) start to where it peak is around 10 minutes as described in Chapter 2. The field *Imbalance* and *Time* are processed like described in Chapter 3.

Source	Symbol	Feature Description	Operation
Local	dt	Local timestamp	Time
Binance	e	Event type	Removed
Binance	E	Event time	Removed
Binance	s	Symbol	Removed
Binance	P	Price change percent	None
Binance	p	Price change	PoC
Binance	w	Weighted average price	PoC
Binance	x	First trade(F)-1 price	PoC
Binance	c	Last price	PoC
Binance	Q	Last quantity	PoC
Binance	b	Best bid price	PoC
Binance	B	Best bid quantity	PoC
Binance	a	Best ask price	PoC
Binance	A	Best ask quantity	PoC
Binance	o	Open price	PoC
Binance	h	High Price	PoC
Binance	l	Low Price	PoC
Binance	v	Base asset volume	PoC
Binance	q	Quote asset volume	PoC
Binance	O	Statistics open time	Removed
Binance	C	Statistics close time	Removed
Binance	F	First trade ID	Removed
Binance	L	Last trade ID	Removed
Binance	n	Total number of trades	PoC
Binance	$ap\_[0-4]$	5x depth - asks price	PoC
Binance	$av\_[0-4]$	5x depth - asks volume	PoC
Binance	$bp\_[0-4]$	5x depth - bids price	PoC
Binance	$av\_[0-4]$	5x depth - asks volume	PoC
Binance	dep	Depth imbalance	Imbalance
ccxt	oc	Aggregated open price	PoC
ccxt	hc	Aggregated high price	PoC
ccxt	lc	Aggregated low price	PoC
ccxt	cc	Aggregated close price	PoC
ccxt	ic	Exchange to market rate	None
CoinMarketCap	cap	Capitalization rate	None

Table 5.1: features that were retrieved and used to train our model.

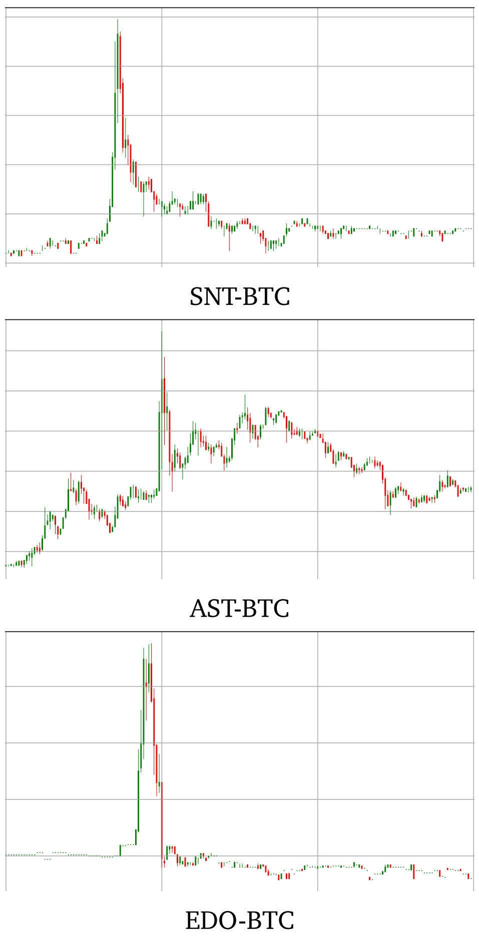
We used the anomaly detection algorithm which we previously described in Chapter 3 to pinpoint P&D intervals. We fetched Open, High, Low, Close, Volume (OHLCV) values that span over the period we collected data, and these OHLCV values had an interval of one hour. The threshold parameters we chose for the algorithm was a price increase of 1.10 and a volume increase of 3.00, and the time lag we chose was 6 hour. By using this algorithm we were able to identify in total 280 anomalies. Of these anomalies we removed 80 that seemed false.

Figure 5.1a shows three P&D anomalies we believe was true, while Figure 5.1b shows three P&D anomalies we believe was false. In Figure 5.1a the price suddenly increases by approximately 20%. This sudden increase only lasts for a few minutes before the price dumps straight down to what it was before the increase. These price characteristics are precisely like the P&D patterns we previously defined in Table 2.3.

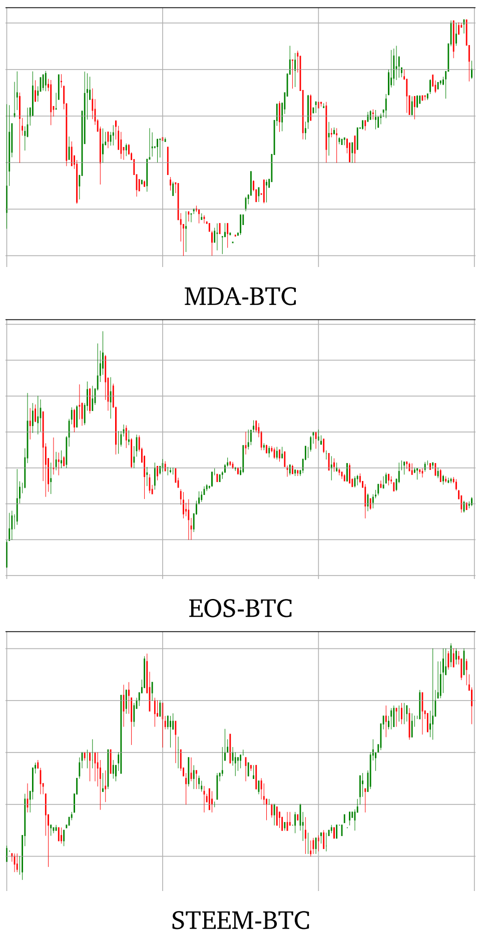
Figure 5.1b on the other hand, show anomalies that we did not believe was P&Ds. And it seems like the price fluctuates substantially, and the price does indeed fluctuate, but the scale in these charts is different from the other. The difference between the lowest and the highest price was around 5% in these charts. If any of these allegedly false P&Ds was true, then these fail in raising the price significantly and suddenly.

To create a dataset, we used the anomalies to label the collected data. The generated dataset was then normalized using the min-max normalization method. We split the dataset into a training set and a test set. The training set consisted of 75% of the dataset, which resulted in data from 104 markets, while the test set contained the remained 25% of the markets, which resulted in 33 markets. The training dataset was undersampled in order to create a balanced dataset.

The model used was a Long Short-term Memory (LSTM) network, where the hidden layer contained a single layer with 50 LSTM cells. Each cell had a time lag of 10 points, which resulted in 50 seconds of memory as the interval between each sample is 5 seconds. We also added dropout to prevent over-fitting as LSTM cells tend to frequently overfit their training data [93]. The output layer contained a single perceptron. The optimizer we used was *adam*, which is an extension to stochastic gradient descent [94]. The optimizer has shown that a model's weights converges faster [95]. The network was trained with the training set over 5000 epochs with the batch size set to 10. To define the performance of the network, we classified all the samples in the test dataset and rounded the probabilistic prediction to either 0 (P&D) or 1 (not P&D).
The computer we used to train our model with had the following specifications:



(a) Anomalies that seemed like a P&D.



(b) Anomalies that not seemed like a P&D.

Figure 5.1: Visualization of 6 out of 280 anomalies that were collected. The three anomalies on the right were later removed as those were believed to be false, while the three on the left was kept as they seemed legitimate.

- CPU - Intel Xeon E5-1620 3.9 GHz
- RAM - 64 GiB DDR3
- GPU - Nvidia GeForce GTX 770

5.2 Results

We use a confusion matrix as the first metric to evaluate the performance of our model. A confusion matrix is structured like Table 5.2 and it shows the number of correct and incorrect classified samples, and it helps us defining further metrics. True positive, tp, is the number of samples that are correctly classified as P&D, while true negative, tn, is the number of samples that are correctly classified as not P&D. fp are samples that are incorrect classified as P&D, and false negatives, fn, are samples that are P&D, but classified as not P&D. p and n are the true total numbers of samples in each class, while p' and n' are the total numbers of predicted samples in each class. Finally, N is the total number of samples.

	Predicted class		
True class	Positive	Negative	Total
Positive	tp: true positive	fn: false negative	p
Negative	fp: false positive	tn: true negative	n
Total	p'	n'	N

Table 5.2: confusion matrix. The diagonal (true) are correctly classified samples. Anti-diagonal (false) are misclassified samples.

As we see in Table 5.3, the test and training datasets are greatly imbalanced. Only 8 821 samples are P&Ds, while 18 321 816 samples are normal trading data. In our test dataset, over one of every 2 000 sample was positive. Table 5.3 contains the aggregated results of every market we tested the model on. The confusion matrix of each market will be presented later in this chapter.

From Table 5.4, we see metrics that can be used to define the performance of our model. The metric *accuracy* and *error*, are contrary measures. Accuracy is the percentage of samples classified correctly, and error is the percentage of samples misclassified. The results of these are 97.82% and 2.17% respectively.

The tp-rate is the ratio of correctly classified P&D samples. When given a positive P&D sample, then there is a 89.67% chance of classifying it correctly.

	Predicted class		
True class	Positive	Negative	Total
Positive	7 910	911	8 821
Negative	388 795	17 933 021	18 321 816
Total	396 705	17 933 932	18 330 637

Table 5.3: Limelight's confusion matrix. In total 17 933 932 samples were correctly classified against 396 705 misclassified samples.

Name	Formula	Result
error	$(fp + fn)/N$	0.0217
accuracy	$(tp + tn)/N$	0.9782
tp-rate	tp/p	0.8967
fp-rate	fp/n	0.0212
sensitivity	tp/p	0.8967
specificity	tn/n	0.9787
recall	tp/p	0.8967
precision	tp/p'	0.0199

Table 5.4: Limelight's performance. The various metrics are definable through using the confusion matrix in Table 5.3.

The fp-rate is similar, when given a negative sample, there is a 2.12% chance of classifying it positively. These two measures are visualized using a Receiver Operating Characteristic (ROC) curve in Figure 5.2. It gives us a visual perspective of the performance of the model. Ideally, a classifier has a tp-rate of 1 and an fp-rate of 0, and a classifier is better the more its curve gets closer to the upper-left corner. The closer the curve is to the diagonal, we make as many true decisions as false ones which is the worst case [96, p. 563]. The Area Under the Curve (AUC) score is the rate of successful classifications [97]. Our ROC curve and AUC shows that our model has accurate predictable capabilities.

The *sensitivity* and *specificity* is the ratio of classifying a positive or negative sample correctly. The sensitivity is the percentage of classifying a positive P&D sample correctly, which is 89.67%. The specificity is the opposite and is the percentage of classifying a negative sample correctly, which is 97.87%.

The metrics shown until now proves that the model has acceptable predictable capabilities. However, the metrics that reveal the flaws of our model are the measures *recall* and *precision*. Recall is precisely like the tp-rate and sensitivity, and is the percentage of classifying a positive sample correctly. Precision on the other hand, is the percentage of samples that are classified as P&D and are correct. In our case 1.99%. So whenever our model predicts a P&D, there is only a 1.99% that it is correct, and this results in quite a few false alarms.

F-score is a common metric that relies on recall and precision. The formula is listed in Figure 5.3. An ideal F-score is 1, and the worst is 0. The measure is flexible in a way that allows us to emphasize recall and precision differently by adjusting the parameter γ. When γ gets closer to 0, the more we emphasize precision, and when γ is 1 we emphasize them equally, and finally when γ is over 1, we emphasize recall more. And as we see, when emphasizing precision most by giving $\gamma = 0.5$, the score yields only 0.0247. When equally

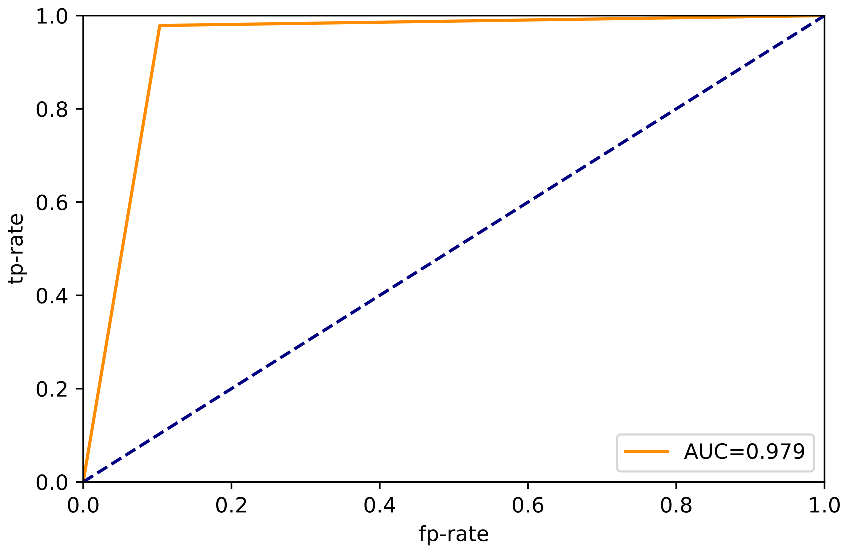


Figure 5.2: Receiver Operating Characteristic describe our classifier performance using a curve. A classifier is better the more its curve gets closer to the upper-left corner. And on the diagonal, it makes as many true decisions as false ones. The Area Under the Curve represents the rate of successfully classifying a sample.

$$F_\beta = (1+\beta^2)\cdot\frac{\text{precision}\cdot\text{recall}}{(\beta^2\cdot\text{precision})+\text{recall}}, \quad \begin{cases} F_{0.5} & = 0.0247 \\ F_1 & = 0.0389 \\ F_2 & = 0.0912 \end{cases}$$

Figure 5.3: F-score

emphasizing them by giving $\gamma = 1$, the score increase slightly, and finally, when emphasizing recall most the score is 0.0912.

Table 5.5 shows the confusion matrix of each classified market. We colorize the cells to visualize the performance of the model. The green cells on the diagonal contain the number of correctly classified samples, and the color intensity illustrates the percentage of correctly classified samples in its class. The red cell on the anti-diagonal consist of the number of misclassified samples and the color intensity represents the percentage of misclassified samples in its class. The intensity in white cells has a split meaning, it is a positive sign on the anti-diagonal (red), but a negative one on the diagonal (green).

WABI-BTC		MITH-BTC		ETH-BTC	
376	36	0	2	0	0
23 440	531 622	26 798	528 675	0	555 474
XRP-BTC		BTT-BTC		OST-BTC	
0	0	0	0	488	53
0	555 474	3 400	552 074	8 705	546 228
CDT-BTC		GNT-BTC		TRX-BTC	
407	51	375	105	0	0
18 303	536 713	20 994	534 000	289	555 185
MTH-BTC		DLT-BTC		SC-BTC	
487	16	584	85	0	0
12 032	542 939	19 223	535 580	7 234	548 240
SNT-BTC		XEM-BTC		VIB-BTC	
323	40	0	0	674	7
8 138	546 973	914	554 559	22 358	532 435
MTL-BTC		HC-BTC		STORM-BTC	
477	36	251	188	0	0
29 687	525 274	5 213	549 822	14 995	540 479
INS-BTC		LUN-BTC		NXS-BTC	
0	1	792	57	0	0
7 032	548 439	12 443	542 182	6 554	548 920
TNB-BTC		NPXS-BTC		ZRX-BTC	
441	3	0	0	0	0
47 144	507 886	3 381	552 093	10 138	545 336
VET-BTC		RCN-BTC		ETC-BTC	
0	0	487	26	160	30
1 321	554 153	10 697	544 264	2 890	552 394
SNM-BTC		SKY-BTC		LOOM-BTC	
1 209	55	141	44	0	0
21 950	532 260	14 423	540 866	1 485	553 989
CVC-BTC		PHX-BTC		PPT-BTC	
238	74	0	1	0	1
8 393	546 769	13 211	542 262	6 010	549 463

Table 5.5: confusion matrices of the 33 tested markets.

5.3 Discussion

To interpret the results, we need to define the potential consequence of misclassifications. So, to push it to the extreme, assume our real job was to classify whether a patient has cancer or not. The model we use to predict cancer has already been trained and does have accurate predictions, but like many other models, it is not entirely flawless, and occasionally it makes an incorrect prognosis of cancer. At some point in the future, the model makes a wrong decision, and it predicts that a patient has cancer, while the patient is actually cancer free. The consequence is that the patient is starting treatment, but it can also have other side effects that impact the patient in a somatic and psychological way, but the patient will continue to live. If we turn the table, and the model predicts that a patient does not have cancer, while the patient truly has. Then, at some point in the future, the patient will die of cancer, which is an unforgivable fault made by the model. This example only clarifies the potential harm of misclassifications, and it is not like that a patient will die from misclassifying P&Ds.

So to apply the consequence of misclassifications in the detection of P&Ds. Assume that, an exchange uses our model to detect P&Ds in order to ban users that participate in them. As we have seen, our model is not perfect, it makes wrong decisions occasionally, and at some point in the future, our model makes a wrong decision and incorrectly predicts a P&D. Subsequently, the exchange could ban a set of investors from using their platform. And by banning investors on false terms the exchange's credibility will be undermined, and have many additional long-term consequences for them. On the other hand, missing P&Ds are not harmful. Despite that, some investors avoid getting banned, but they might not be so lucky if they ever try again.

From the opposing point of view, the consequence of misclassifications for investors are slightly different from exchanges that seek to prevent P&Ds. As investors incentive is to buy early in a P&D and sell when the coin peak. And if they buy on false terms, it is not catastrophic; they will not lose their assets. Most likely, there was only a small increase in the price that made the model issue a false P&D, which can actually prove to be profitable. Also, occasionally missing a P&D is not critical, it only results in being passive during a P&D, and one does not lose assets of being passive, but one does skip out on potential profit.

As we saw from our results, all the metrics except the F-score and precision performs excellent and surpasses other models that detect P&Ds. However, The F-score and precision illustrate a fatal flaw in our model, and as the model currently stands, deploying it will be remarkably hard as there would be too many false alarms. There are ways to adjust the precision but at a cost.

Improving it will with a high likelihood cost us our current tp-rate and make that worse, but due to the nature of the metrics, increasing precision will also increase other measures. As changing the precision either increase the true positives or decrease false positives.

Increasing true positives in our case can make precision worse. We must try to make the model fit to additional positive samples simultaneously, and there are over 2 000 more negative samples. So, our angle should be to decrease the false positives, in order to increase precision, but that may result in a decrease of true positives.

To adjust the false positives, we can be more strict when classifying, as mentioned in end of Section 5.1, we rounded each prediction. Probabilities over 0.5 is classified as negative, while probabilities under 0.5 are classified as positive. If the prediction is precisely on the 0.5 threshold, then the model can not separate it, but these cases are scarce, and a case that is often entirely ignored. However, the point is that we can adjust this 0.5 threshold in terms of how strict we want to be. By lowering this threshold, we can be more strict; we can be more sure of whether it genuinely is a P&D or not. This method is also very flexible as the threshold is simple to adjust and do not require us to train our model again.

Other methods to adjust false positives is to train our model with an imbalanced dataset that inclines the negative class. This method is not preferred as it requires us to retrain our model over and over again to evaluate it. This method was also our first trial, which when evaluated resulted in 99.99% accuracy. However, there was only one problem. It could not recognize any P&Ds; it classified all samples as negative and had a tp-rate of 0%.

We also tried to adjust the loss-cost of each class during training, such that it favored positive over negative. This method was error-prone and incredibly hard to control as we had to guess how much our model's weights should change during training.

Ultimately, before using Limelight, one should consider the various consequences of misclassifying and then adjust the decision threshold accordingly to minimize misclassifications in one class.

6 Conclusion

Since the emergence of cryptocurrencies, exchanges have and still play a central role in the cryptocurrency ecosystem. With over 99% of all trades happening on exchanges, one can say that they are an appealing platform for investors despite that they are mostly unregulated. Unregulated investment platforms attract scammers who organize various price manipulation schemes like Pump-and-Dumps (P&Ds), and this have become troublesome on exchanges as the scammers produce in total $7 million in daily trading volume.

Our thesis statement was to detect P&Ds in real-time by using deep learning, and we did so by designing and implementing Limelight. We modeled it like a Machine Learning (ML) pipeline, where the first stage collected data from Binance, CoinMarketCap, and aggregated data from Ccxt. Then, with the collected data we defined a new dataset by cleaning and engineering new features of it. To label our dataset we used an anomaly detection algorithm to pinpoint P&Ds, and to reduce the occurrences of false P&Ds we manually removed them. With a labeled dataset, we normalized it, and trained a Long Short-term Memory (LSTM) network.

To train our LSTM network, we collected data over a period of 33 days resulting in total 47 GB of data. The anomaly detection algorithm pinpointed 280 anomalies over the period we collected data, and from these anomalies we removed 80 false P&Ds. The LSTM network we trained has an accuracy of 97.82% and an Area Under the Curve (AUC) of 0.979.

6.1 Contributions

We published two packages on PyPi, `CoinMarketCapAPI` and `timeseries`. The former extracts data from the cryptocurrency tracking site CoinMarketCap, and the latter works with time series data. Both modules can be easily installed with PyPi's package install pip. There are unit tests for both modules, and integration tests for the CoinMarketCapAPI module as this module have multiple dependencies. We used `pytest`, a testing framework in Python, to execute all the tests and calculate the percentage of code that was covered. In the timeseries package, we were able to cover 100% of the module, while in the CoinMarketCapAPI, we were able to cover 86%. The following tables show code statistics regarding Limelight, CoinMarketCapAPI, and timeseries.

The following tables shows lines of code in each project, and it was calculated by using a tool called `loc`. Noticeably, docstrings in Python is counted as code lines and not as a comments.

Language	Files	Lines	Blank	Comment	Code
Python	13	2 371	383	26	1 962
Markdown	1	108	27	0	81
Makefile	1	35	10	0	25
Text	1	3	0	0	3
Sum	16	2 517	420	26	2 071

Table 6.1: Lines of code - CoinMarketCap module

Language	Files	Lines	Blank	Comment	Code
Python	5	207	48	6	153
Markdown	1	48	13	0	35
Makefile	1	29	8	0	21
Sum	7	284	69	6	209

Table 6.2: Lines of code - Timeseries module

Language	Files	Lines	Blank	Comment	Code
Python	39	2 026	377	156	1 493
Makefile	1	32	9	0	23
JSON	1	10	0	0	10
Text	1	3	0	0	3
Markdown	1	1	0	0	1
Sum	43	2 072	386	156	1 530

Table 6.3: Lines of code - Limelight

Dataset

Another notable contribution is the two datasets we have produced during the thesis. The first dataset contains raw data that are directly extracted from Binance, CoinMarketCap, and aggregated data from multiple sources, and is available at https://bit.ly/2Wpkfws. The second dataset, the one we used to train our LSTM network and produce the presented results, is available at https://bit.ly/2ExoGYJ. Hopefully, these two datasets becomes useful for researchers that also seeks to detect P&D events.

6.2 Concluding Remark

Through designing, implementing, and evaluating Limelight, we prove that it is feasible to detect P&Ds in real-time using deep learning. With an accuracy of almost 98% the model achieves a very high accuracy. Unfortunately the imbalance in the datasets gives us a low precision of 0.2%, but the consequences of mispredicting is not necessarily negative.

Besides creating Limelight, we implemented two other Python modules that we published on PyPI, and shared two datasets that can be further used in the detection of P&Ds.

6.3 Future Work

Improvement Suggestions

Labeling P&D samples in the dataset, is and will remain problematic as long as we do not have any prior knowledge regarding whether a P&D actually happened or not. Optimally, knowing every P&D event that happen during the phase where Limelight collects data allows us to label the samples with a higher precision, which results in more accurate results. Hence, if some publish accurate and detailed P&D events, or if we improve the anomaly detection such that we get more accurate measures, then we can also be more precise when labeling samples.

In this thesis, Limelight only uses a few features from CoinMarketCap. We only used those features we genuinely believed is useful in detecting P&D. Since this is not a problem that can be easily solved, more features may improve the performance of the model. Thus, we can utilize more data from CoinMarketCap in the future. The same can be applied to the cryptocurrency source that aggregate data from multiple sources, we can probably extract more data from it than just Open, High, Low, Close (OHLC) values.

A typical preprocessing stage is to reduce the number of dimensions in the data. Because of the huge class imbalance and that samples are timeseries data, we have to be careful when reducing dimensions. A method that we believe is optimal when reducing dimensions is a method called backward selection.

Applications

We see a crossroad when it comes to usage of Limelight, either be ethical or unethical. Exchanges can be ethical in terms of preventing P&Ds by allowing Limelight to detect them in real-time and punish or blacklist those investors that participate them. Such that their investors are buying and selling assets on the same terms, instead of a few investors deceiving others and profit from them.

Investors can both be ethical or unethical. They can decide to participate in P&Ds by buying assets early in the P&D and sell when the coin peak, however, this requires Limelight to detect them very early in order to profit from this technique. Investors can also use Limelight ethically by preventing investing when a P&D occurs, such that they do not lose any assets, but not profiting from it either.

Bibliography

[1] J. Kamps and B. Kleinberg, “To the moon: Defining and detecting cryptocurrency pump-and-dumps,” *Crime Science*, vol. 7, no. 1, p. 18, 2018.

[2] (Jun. 1, 2014). Definition of fiat money, Lexicon, [Online]. Available: http://lexicon.ft.com/Term?term=fiat-money (visited on 12/06/2018).

[3] A. Sotiropoulou and D. Guégan, “Bitcoin and the challenges for financial regulation,” *Capital Markets Law Journal*, vol. 12, pp. 466–479, Sep. 14, 2017.

[4] (Oct. 13, 2018). Decentralized crypto exchanges vs centralized exchanges like binance, bittrex, Coinsutra, [Online]. Available: https://coinsutra.com/decentralized-vs-centralized-crypto-exchange/ (visited on 03/17/2019).

[5] J. Xu and B. Livshits, “The anatomy of a cryptocurrency pump-and-dump scheme,” *arXiv preprint arXiv:1811.10109*, 2018.

[6] T. Lo, D. Shin, and B. Wang, “Cryptocurrency pump-and-dump schemes,” Feb. 2019.

[7] (Feb. 19, 2019). Why pump and dump won’t work with anchor? ANCHOR, [Online]. Available: https://theanchor.io/why-pump-and-dump-wont-work-with-anchor/ (visited on 04/28/2019).

[8] C. Ramos and N. Golub, “Cryptocurrency pumping predictions: A novel approach to identifying pump and dump schemes,” 2017.

[9] “Amazon machine learning - developer guide,” 2019.

[10] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.

[11] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, “A deep learning approach for network intrusion detection system,” pp. 21–26, 2016.

[12] H.-K. Peng and R. Marculescu, “Multi-scale compositionality: Identifying the compositional structures of social dynamics using deep learning,” *PloS one*, vol. 10, no. 4, e0118309, 2015.

[13] P. j. Denning, D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young, “Computing as a discipline,” *Communication of the ACM*, vol. 32, pp. 9–23, Jan. 1989.

[14] P. Halvorsen, S. Sægrov, A. Mortensen, D. K. Kristensen, A. Eichhorn, M. Stenhaug, S. Dahl, H. K. Stensland, V. R. Gaddam, C. Griwodz, *et al.*, “Bagadus: An integrated system for arena sports analytics: A soccer case

study," in *Proceedings of the 4th ACM Multimedia Systems Conference*, ACM, 2013, pp. 48–59.

[15] D. Johansen, M. Stenhaug, R. B. Hansen, A. Christensen, and P.-M. Høgmo, "Muithu: Smaller footprint, potentially larger imprint," in *Seventh International Conference on Digital Information Management (ICDIM 2012)*, IEEE, 2012, pp. 205–214.

[16] E. Tedeschi, "Trading network performance for cash in the bitcoin blockchain," Master's thesis, UiT Norges arktiske universitet, 2017.

[17] J. F. Mikalsen, "Firechain: An efficient blockchain protocol using secure gossip," Master's thesis, UiT Norges arktiske universitet, 2018.

[18] H. D. Johansen, R. V. Renesse, Y. Vigfusson, and D. Johansen, "Fireflies: A secure and scalable membership and gossip service," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 2, p. 5, 2015.

[19] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.

[20] (Sep. 14, 2018). Python programming language ditches 'master-slave' terms, pissing off some, Gizmodo, [Online]. Available: https://gizmodo.com/python-programming-language-ditches-master-slave-terms-1829057867 (visited on 05/01/2019).

[21] (Sep. 13, 2018). Redis does a python, crushes 'offensive' master, slave code terms, The Register, [Online]. Available: https://www.theregister.co.uk/2018/09/13/redis_master_slave/ (visited on 05/01/2019).

[22] (Mar. 14, 2016). Let's stop saying master/slave, Medium, [Online]. Available: https://medium.com/@mikebroberts/let-s-stop-saying-master-slave-10f1d1bf34df (visited on 05/01/2019).

[23] N. Baychenko, "Implementing a master/slave architecture for a data synchronization service," 2018.

[24] (Jan. 16, 2014). Sharding, scaling, data storage methodologies, and more: Insights on big data, DZone, [Online]. Available: https://dzone.com/articles/sharding-scaling-data-storage (visited on 05/01/2019).

[25] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[26] (May 2, 2019). Benefits of pub/sub messaging, Amazon, [Online]. Available: https://aws.amazon.com/pub-sub-messaging/benefits/ (visited on 05/21/2019).

[27] (Mar. 12, 2013). Design patterns: Pubsub explained, CodeKraft, [Online]. Available: https://abdulapopoola.com/2013/03/12/design-patterns-pub-sub-explained/ (visited on 05/21/2019).

[28] C. V. Ramamoorthy and H. F. Li, "Pipeline architecture," *ACM Computing Surveys (CSUR)*, vol. 9, no. 1, pp. 61–102, 1977.

[29] (Nov. 15, 2018). Guide to building machine learning pipeline architecture, Xenonstack, [Online]. Available: https://www.xenonstack.com/insights/what-is-a-machine-learning-pipeline/ (visited on 03/27/2019).

[30] (Feb. 19, 2019). Cryptocurrency, Investopedia, [Online]. Available: https://www.investopedia.com/terms/c/cryptocurrency.asp (visited on 02/20/2019).

[31] (Feb. 20, 2019). Top 100 cryptocurrencies by market capitalization, CoinMarketCap, [Online]. Available: https://coinmarketcap.com/ (visited on 02/20/2019).

[32] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[33] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, pp. 22–23, 2013.

[34] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. Princeton University Press, 2016, ISBN: 978-0-691-17169-2.

[35] M. Möser and R. Böhme, "Trends, tips, tolls: A longitudinal study of bitcoin transaction fees," *International Financial Cryptography Association 2015*, Jan. 1, 2015.

[36] P. R. Rizun, "A transaction fee market exists without a block size limit," Aug. 4, 2015.

[37] (Dec. 3, 2018). Bitcoin energy consumption index, Digiconomist, [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption (visited on 12/03/2018).

[38] (Dec. 12, 2017). How long do bitcoin transactions take? Coin Central, [Online]. Available: https://coincentral.com/how-long-do-bitcoin-transfers-take/ (visited on 03/13/2019).

[39] (Jan. 1, 2019). Bitcoin confirmations, Buy Bitcoin Worldwide, [Online]. Available: https://www.buybitcoinworldwide.com/confirmations/ (visited on 03/13/2019).

[40] (Oct. 2018). Cryptocurrency exchanges and custody providers: International regulatory developments, Norton Rose Fulbright, [Online]. Available: https://www.nortonrosefulbright.com/en/knowledge/publications/e383ade6/cryptocurrency-exchanges-and-custody-providers-international-regulatory-developments (visited on 03/13/2019).

[41] (May 9, 2019). Bid-ask spread, Investopedia, [Online]. Available: https://www.investopedia.com/terms/b/bid-askspread.asp (visited on 05/30/2019).

[42] (Jan. 20, 2010). Beware of these 9 cryptocurrency scams, HackerNoon, [Online]. Available: https://hackernoon.com/beware-of-these-cryptocurrency-scams-f49ef9b18a5d (visited on 03/13/2019).

[43] (Dec. 16, 2018). Wash trading: How crypto exchanges are faking 67% of trade volume, Blog Cotten, [Online]. Available: https://blog.cotten.io/wash-trading-how-crypto-exchanges-are-faking-67-of-trade-volume-19ba61cbad4 (visited on 03/14/2019).

[44] (Jan. 2019). Best cryptocurrency exchanges: In-depth review 2019, Toshi Times, [Online]. Available: https://toshitimes.com/best-cryptocurrency-exchange-2019/ (visited on 03/13/2019).

[45] (Feb. 27, 2019). 10 best cryptocurrency exchanges for crypto trading in 2019, Coin Switch, [Online]. Available: https://coinswitch.co/news/10-best-cryptocurrency-exchanges-for-crypto-trading-in-2019 (visited on 03/13/2019).

[46] (Mar. 13, 2019). Cryptocurrency exchanges by trade volume (page 3), CoinMarketCap, [Online]. Available: https://coinmarketcap.com/rankings/exchanges/3 (visited on 03/13/2019).

[47] (Jan. 28, 2019). Crypto startup wants you to trade on exchanges without trusting them, Coindesk, [Online]. Available: https://www.coindesk.com/crypto-startup-wants-you-to-trade-on-exchanges-without-trusting-them (visited on 03/14/2019).

[48] (Jun. 10, 2018). What is api trading and how is it applied to crypto? Brave Newcoin, [Online]. Available: https://bravenewcoin.com/insights/what-is-api-trading-and-how-is-it-applied-to-crypto (visited on 03/15/2019).

[49] (Feb. 12, 2019). Depth of market – dom definition, Investopdia, [Online]. Available: https://www.investopedia.com/terms/d/depth-of-market.asp (visited on 03/06/2019).

[50] (Mar. 6, 2019). Order book, Investopdia, [Online]. Available: https://www.investopedia.com/terms/o/order-book.asp (visited on 03/11/2019).

[51] (Aug. 6, 2018). How to read an exchange order book? Coincodex, [Online]. Available: https://coincodex.com/article/2236/how-to-read-an-exchange-order-book/ (visited on 03/06/2019).

[52] (Dec. 5, 2018). Mit: Crypto pump-and-dumps see $7 million in volume every day, Cryptocurrency News, [Online]. Available: https://www.ccn.com/mit-crypto-pump-and-dumps-see-7-million-in-volume-every-day/ (visited on 01/15/2019).

[53] (Nov. 24, 2018). Pump and dump in crypto: Cases, measures, warnings, Cointelegraph, [Online]. Available: https://cointelegraph.com/news/pump-and-dump-in-crypto-cases-measures-warnings (visited on 01/15/2019).

[54] T. Bouraoui, "Stock spams:an empirical study on penny stock market," *International Review of Business Research Papers*, vol. 5, pp. 292–305, Jun. 2009.

[55] S. Temple, "Cybertrading: Financial markets and the internet," *Australian Law Librarian*, vol. 8, pp. 337–345, 2000.

[56] (Jan. 23, 2018). Inside the group chats where people pump and dump cryptocurrency, The Outline, [Online]. Available: https://theoutline.com/post/3074/inside-the-group-chats-where-people-pump-and-dump-cryptocurrency?zd=1&zi=n3jct6pm (visited on 01/15/2019).

[57] M. Bartoletti, S. Carta, T. Cimoli, and R. Saia, "Dissecting ponzi schemes on ethereum:identification, analysis, and impact," Mar. 10, 2017.

[58] M. Voets, “Deep learning: From data extraction to large-scale analysis,” 2018.

[59] R. Chalapathy and S. Chawla, “Deep learning for anomaly detection: A survey,” *arXiv preprint arXiv:1901.03407*, 2019.

[60] (Apr. 2019). E-handbook of statistical methods, NIST/SEMATECH, [Online]. Available: http://www.itl.nist.gov/div898/handbook/ (visited on 04/19/2019).

[61] (Aug. 27, 2015). Understanding lstm networks, colah’s blog, [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 05/06/2019).

[62] K. J. Hunt, D. Sbarbaro, R. Żbikowski, and P. J. Gawthrop, “Neural networks for control systems—a survey,” *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.

[63] Z. C. Lipton, J. Berkowitz, and C. Elkan, “A critical review of recurrent neural networks for sequence learning,” *arXiv preprint arXiv:1506.00019*, 2015.

[64] Y. Bengio, P. Simard, P. Frasconi, *et al.*, “Learning long-term dependencies with gradient descent is difficult,” *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[65] (Aug. 9, 2016). A quick introduction to neural networks, the data science blog, [Online]. Available: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/ (visited on 05/07/2019).

[66] (Aug. 28, 2013). Multi-layer neural network, Stanford, [Online]. Available: %5Curl%7Bhttp://deeplearning.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/%7D (visited on 05/07/2019).

[67] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[68] K. Hornik, “Some new results on neural network approximation,” *Neural networks*, vol. 6, no. 8, pp. 1069–1072, 1993.

[69] T. Leangarun, P. Tangamchit, and S. Thajchayapong, “Stock price manipulation detection using a computational neural network model,” *international conference on advanced computational intelligence*, Feb. 14, 2016.

[70] (Dec. 9, 2018). Study: Pump and dump schemes account for $7 million in monthly volume, UTB Cryptocurrency News, [Online]. Available: https://usethebitcoin.com/study-pump-and-dump-schemes-account-for-7-million-in-monthly-volume/ (visited on 05/01/2019).

[71] (Sep. 5, 2018). How to build a better machine learning pipeline, datanami, [Online]. Available: https://www.datanami.com/2018/09/05/how-to-build-a-better-machine-learning-pipeline/ (visited on 03/28/2019).

[72] S. Zhang, C. Zhang, and Q. Yang, “Data preparation for data mining,” *Applied artificial intelligence*, vol. 17, no. 5-6, pp. 375–381, 2003.

[73] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *Data Engineering*, vol. 23, pp. 3–14, 2000.

[74] (Jun. 11, 2016). Linear interpolation with excel, Dagraa, [Online]. Available: https://www.datadigitization.com/dagra-in-action/linear-interpolation-with-excel/ (visited on 05/02/2019).

[75] (Dec. 10, 2018). Getting data ready for modelling: Feature engineering, feature selection, dimension reduction (part 1), Towards Data Science, [Online]. Available: https://towardsdatascience.com/getting-data-ready-for-modelling-feature-engineering-feature-selection-dimension-reduction-77f2b9fadc0b (visited on 05/02/2019).

[76] (Apr. 13, 2018). Pump and dump schemes: How they work in cryptocurrency, Blockonomi, [Online]. Available: https://blockonomi.com/pump-and-dump/ (visited on 03/16/2019).

[77] B. Frénay and A. Kabán, "A comprehensive introduction to label noise," *European Symposium on Artificial Neural Network*, pp. 667–676, Apr. 23, 2014.

[78] (Mar. 6, 2019). Sliding time windows, Redhat, [Online]. Available: https://access.redhat.com/documentation/en-us/jboss_enterprise_brms_platform/5/html/brms_complex_event_processing_guide/sect-sliding_time_windows (visited on 03/17/2019).

[79] V. A. Siris and F. Papagalou, "Application of anomaly detection algorithms for detecting syn flooding attacks," *Computer Communication*, vol. 29, pp. 1433–1442, 2006.

[80] M. Grill, T. Pevny, and M. Rehak, "Reducing false positives of network anomaly detection by local adaptive multivariate smoothing," *Journal of Computer and System Sciences*, vol. 83, no. 1, pp. 43–57, 2017.

[81] (May 8, 2019). Ethereum, CoinMarketCap, [Online]. Available: https://coinmarketcap.com/currencies/ethereum/ (visited on 05/08/2019).

[82] (May 6, 2019). Normalize data, Microsoft Azure, [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data (visited on 05/09/2019).

[83] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Transactions on nuclear science*, vol. 44, no. 3, pp. 1464–1468, 1997.

[84] (Mar. 26, 2019). Normalizataion, Google Machine Learning, [Online]. Available: https://developers.google.com/machine-learning/data-prep/transform/normalization (visited on 05/09/2019).

[85] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle, "On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 891–927, 2016.

[86] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PloS one*, vol. 11, no. 4, e0152173, 2016.

[87] S. K. Panda, S. Nag, and P. K. Jana, “A smoothing based task scheduling algorithm for heterogeneous multi-cloud environment,” in *2014 International Conference on Parallel, Distributed and Grid Computing*, IEEE, 2014, pp. 62–67.

[88] (May 26, 2018). Deep learning unbalanced training data?solve it like this., Towards Data Science, [Online]. Available: %5Curl%7Bhttps://towardsdatascience.com/deep-learning-unbalanced-training-data-solve-it-like-this-6c528e9efea6%7D (visited on 05/10/2019).

[89] (Nov. 19, 2018). Handling imbalanced datasets in deep learning, Towards Data Science, [Online]. Available: https://towardsdatascience.com/handling-imbalanced-datasets-in-deep-learning-f48407a0e758 (visited on 05/10/2019).

[90] (May 2019). Popularity of programming language, PYPL, [Online]. Available: https://pypl.github.io/PYPL.html (visited on 05/11/2019).

[91] (Aug. 2, 2017). Global interpreter lock, Python, [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock (visited on 05/14/2019).

[92] (Nov. 13, 2018). Web socket streams for binance, Github, [Online]. Available: https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md (visited on 05/14/2019).

[93] (May 5, 2017). Gentle introduction to the adam optimization algorithm for deep learning, Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/use-weight-regularization-lstm-networks-time-series-forecasting/ (visited on 05/19/2019).

[94] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[95] (Jul. 3, 2017). Gentle introduction to the adam optimization algorithm for deep learning, Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (visited on 05/18/2019).

[96] E. Alpaydin, *Introduction to Machine Learning*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2014, ISBN: 9780262028189. [Online]. Available: https://books.google.no/books?id=NP5bBAAAQBAJ.

[97] A. P. Bradley, “The use of the area under the roc curve in the evaluation of machine learning algorithms,” *Pattern recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.