



Department of Computer Science

# **A Computational Stylometry Analysis on Underground Cryptomarkets To Find Doppelganger Accounts**

Anan Shekher Srivastava

---

A dissertation submitted to the University of Bristol in accordance with the requirements of the  
degree of Master of Science in the Faculty of Engineering

---

September 2020 | COMSM3100-20

# **Declaration:**

This dissertation is submitted to the University of Bristol per the requirements of the degree of Master of Science in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Anan Shekher Srivastava, September 2020

## Executive Summary

Underground cryptomarkets form a part of the dark web that has gained rapid popularity over the last few years. These markets are used by cybercriminals to trade illegal products and perform illegal activities such as trafficking, drugs, selling weapons etc. Such marketplaces provide a haven for these criminals as browsers such as TOR mask multiple layers of IP addresses that protects the identity of these criminals. While agencies such as FBI and Europol have conducted operations to catch these criminals and shut down these markets, evidence has shown it has had little to no effect in the businesses of these markets and their users. Operation Onymous in 2014 shut down *Silk Road 2.0* which was the biggest underground marketplace. It was expected that this would create unrest amongst the users of the dark web, but the exact opposite happened! The users migrated to other smaller marketplaces such as *Abraxus* and the news of this in the media attracted more people to the dark web and informed them about their existence. A better solution to this problem was needed. Hence, this project drew motivation from that and aimed to find a more effective way to track down these criminals.

This project investigates 2 things:

1. How a technique called stylometry, can be applied to textual data from these markets to identify authors of anonymous text, across forums, and form associations between them.
2. Further, this project argues why standard data cleaning methods are insufficient when dealing with data from such cryptomarkets and provide a meaningful data parsing methodology.

Using Natural Language Processing (NLP) techniques along with Neural Networks and other classifiers, we were successfully able to achieve both objectives as mentioned in Section 1.1. We worked with 14 different machine learning models as discussed in the Results Section, and concluded that LinearSVC worked best for our dataset. This project successfully achieves –

- Introducing a new parsing algorithm based on a specific range of words per user per text. This was done to avoid the problem of having an imbalanced dataset. The algorithm also includes some of the common cleaning methods and motivates why that is not enough to deal with data of this sort. We discuss this in-depth in Section 3.4.
- A tool written in Python that applies Stylometry to find common patterns between the writing styles of these users and form association based on the text. It classifies – given two pieces of text, what is the probability that it is written by the same person.

This study is important to appreciate as the charm of these markets in terms of money attracts many people, especially children and young. The owner of Silk Road made \$80 million in 10 months. The idea of making millions in the span of a year or two is lucrative to people who come from low-class neighborhoods. However, once people start dealing with these markets, it is very hard for them to get out and they forever lead a life on the run. This project hopes to avoid that.

## Acknowledgement

This project would not have been a success without the expert guidance of my thesis supervisor, Dr Matthew Edwards. I would like to thank him for his continued support right from project selection to project closure. His help, patient guidance and valued feedbacks helped me improve the overall outcome of this project, both qualitatively and quantitatively.

I would also like to thank my other subject professors and colleagues that made my University of Bristol journey more meaningful and impactful.

# Contents

Executive Summary .....	3
Acknowledgement .....	4
1. Introduction .....	7
1.1. Objectives .....	9
1.2. Added Value .....	9
1.3. Deliverables.....	9
2. Literature Review .....	12
2.1. Contextual Background .....	13
2.1.1. Underground Cryptomarkets.....	13
2.2. Related Studies.....	16
2.2.1. Stylometry.....	18
2.2.2. Author Attribution.....	19
2.3. Technical Background.....	20
2.3.1. Stylometric techniques.....	21
2.3.2. Oversampling & Under sampling .....	22
2.3.3. TFIDF Vectorizer .....	24
3. Project Implementation.....	25
3.1. Hardware and Software Tools .....	25
3.2. Development Decisions .....	26
3.2.1. Scripting Language Selection .....	27
3.2.2. Environment.....	27
3.2.3. Dataset.....	28
3.2.4. Efficiency .....	32
3.2.5. Clean Code .....	33
3.3. Clean Up Strategies .....	34
3.3.1. Removing unwanted characters .....	34
3.3.2. Tokenization.....	35
3.3.3. Removing Stop Words .....	35
3.3.4. Lemmatization/Stemming .....	36
3.4. Algorithm Selection .....	36

3.4.1.	MultinomialNB .....	43
3.4.2.	Logistic Regression .....	44
3.4.3.	LinearSVC .....	44
3.4.4.	Passive Aggressive Classifier .....	44
3.4.5.	Neural Networks model 1 .....	45
3.4.6.	Neural Networks Model 2.....	46
3.4.7.	Neural Networks Model 3.....	47
3.4.8.	Neural Networks Model 4.....	48
3.4.9.	Neural Network Model 5 .....	49
3.4.10.	Pretrained Glove Embedding .....	50
4.	Results.....	52
4.1.	Pre clean up.....	52
4.1.1.	Classification Report.....	52
4.2.	Post clean up.....	54
4.2.1.	Results – Post cleanup.....	54
4.3.	Summary.....	54
4.4.	Critical Evaluation.....	55
4.4.1.	Uniqueness.....	55
5.	Conclusion.....	57
5.1.	Scope Extension .....	57
5.1.1.	Author Obfuscation. ....	57
5.1.2.	Factors that Influence User Decisions.....	58
5.1.3.	Incorporating datasets from languages other than English .....	58
5.2.	Limitations.....	58
5.3.	Future Works.....	58
	References .....	59

## 1. Introduction

This research looks at applying stylometric techniques to underground cryptomarkets, to recognize users with multiple accounts, based on their writing style.

Underground markets form a safe place for people who trade in illegal activities. There has been a sharp increase in the number of people that have signed up on marketplaces such as *Alphabay* and *Abraxus*. With the increased cash flow into these places, they have grown bigger and stronger. The amount of money that people make on such sites, unfortunately, attracts many young people from low-class neighborhoods. The lavish lifestyle lures people with financial difficulties to join these markets.

People have been known to create multiple accounts on these underground places so as to increase personal income, albeit through illegal ways. Vendors usually create multiple accounts with different alias names on different sites. This not only helps them protect their identity but also increases their chances of being the vendor of the choice. For the simple fact, more the accounts, more the chances of doing business. While this may seem like a great strategy, it also leaves room for mistakes.

Law enforcement agencies like the DEA and FBI use this to their advantage to scrape through the data to find a potential address or a phone number. Having multiple accounts increases the chances of finding something that may help them lead to these cybercriminals. One unique way to find these criminals is to find out how many accounts they have. This is done to map different accounts to the same person. There are many techniques to do this. One such technique is Stylometry.

Stylometry can be used to identify a pattern in the way people write. We discuss an example of this in the Literature Review section about how stylometry was used to identify that Robert Galbraith was JK Rowling. It can be used to match two or more pieces of textual data and return a probability of how likely they are written by the same person. This technique is particularly helpful with plagiarism detection, code stylometry and blog plagiarism. Some have even used it to find plagiarism in musical notes and lyrics.

While there has been some previous research in the field of linguistic stylometry, as we thoroughly discuss in the Literature Review section, it remains an area to be investigated. Particularly, linguistic stylometry has rarely been used in a dataset from the underground cryptomarkets. It is only now that law enforcement agencies have started to realize how big these markets are and what impact they have on people, especially young, and society. Few researchers that have used stylometry in underground cryptomarkets have used it on meta-data obtained from a publicly available dataset. Further, previous researchers have suffered having inadequate real data. They have been known to create artificial accounts to create synthetic datasets. In our project, we had the luxury of working with about 1.2 million rows of real data comprising of the text, username, date, and PGP (Pretty Good Privacy) keys.

As studies [7] have revealed that the recent trend in the rise of cashflow in the underground cryptomarkets has increased multiple folds, finding these criminals and putting an end to these markets has become of paramount importance. In this project, we aim to develop a data cleaning algorithm (objective 2) specific to the data of these markets (data from these markets include texts in many languages, short forms, slangs etc.) that can clean and effectively remove *noise* from the data. The algorithm we propose can be used to analyze the chats from these markets that might reveal key information about users' buying patterns, shipping addresses, connections within these communities etc. Further, we have developed our algorithm in such a way that all the data from multiple markets can be parsed at once, as we were dealing with data from 77 different underground markets. We elaborate more on this in the Literature Review and Project Implementation section.

We also look at how stylometry can be applied to these underground marketplaces (objective 1). For example, the language used in these forums are a combination of illegal characters, slang and capped words. Regular cleaning techniques might not be enough to clean a dataset of this sort and we wanted to explore ways to do this. Our key contributions are mentioned in Section 1.3.

This study is important to conduct as these markets are involved in activities like child trafficking and releasing private information of people. Some reports even suggest that people may use these markets to hire assassins. Further, it is imperative to find and stop these criminals as many young people get influenced by these activities and are never able to get out. Once someone is involved in such activities, they become a criminal in the eyes of the law and always lead a life on the run. Hideouts become a safe place. Unfortunately, no matter how strong or power a cryptomarket maybe, no matter how well the vendors and buyers try to hide, they eventually get caught and are imprisoned for most of their life. It also negatively impacts the economy of the country and their families.

In this project, we discuss some of the secret operations that have been conducted before and to shut down these markets and their impact. We discuss operations like Operation Onymous [1] [4], that saw the shutdown of Silk Road, probably one of the biggest marketplaces to ever have existed. Silk Road 1.0 and Silk Road 2.0 were an underground dark marketplace that was best recognized as the online 'drug cartel'. Silk Road 1.0 was launched in 2011 and soon rose to the top spot. It traded illegal drugs, weapons, and personal data of people. After its shutdown in 2013, it was expected that it would be a major setback for buyers and vendors. However, all the media publicity made people more aware of the existence of such marketplace which sprung the launch of Silk Road 2.0 in 2014. However, just after 10 months of launching the site, the owner was arrested, and the site was shut down. The important point to note here is that while the site remained active for only 10 months, it earned a little over 80 million USD [8].

This study will provide an algorithm to parse data from such forums and show how to apply stylometry to underground marketplaces.



## 1.1. Objectives

### 1. Apply stylometry in underground cryptomarkets

This project aims to investigate how stylometry can be used in underground cryptomarkets to identify and match authors of anonymous text. That is, “given the forum posts of two users, A and B, are they or are they not written by the same user?” In other words: is B an alias of A?

### 2. Introduce a robust algorithm to parse data from underground cryptomarkets

Since the data from underground cryptomarkets is very different (translated text, short messages, multiple languages) from the data from other more general sources (Reddit, Stack Overflow), the general data pre-processing techniques are not enough to clean the data. This project aims to introduce a general robust algorithm that will successfully parse and clean this type of data – before using it for model training.

This project argues the statement: Writing style and textual hints expressed in language can be quantified and characterized, thus can be used to link authors.

## 1.2. Added Value

The goal of this project is to understand how stylometry can be used to link users with multiple accounts just by analyzing their text. We have seen an increase in this field of research as concerns mount over the impact of these illegal activities on the day to day lives of the people involved, economies and cybersecurity of people. Previous attempts have failed to deal with large, real-world datasets. This project deals with actual data and actual chats between users of these marketplaces. It will fill the research gap there in the sense that albeit increasing, there has been limited research in this field. Further, based on the result of the study, this method and parsing algorithm can be used at a large scale to find links between accounts as we rightly match these accounts for 80% of the time. We discuss more on this in the Results section. Further, we also introduce a parsing algorithm that combines huge datasets and cleans them more efficiently than the standard cleaning techniques out there. In a sentence, this project could help law enforcement agencies with their mission to identify these criminals.

## 1.3. Deliverables

Below are the outputs from this project:

- I. A dataset of 1.18 million linked accounts distributed between 1959 unique people who have multiple fake accounts on different underground websites, and 234000 non-linked accounts between forum contributors on 77 underground cryptomarkets, containing posts and timestamps for each user's contribution. This dataset was extracted from an existing collection [9]. Linked accounts were identified through PGP keys in common between different users on different forums. This dataset can be used to train and evaluate stylometric classifiers for re-identifying offenders between marketplaces. The advantage of this dataset over previous datasets used by previous

researchers is that none of the entries was artificially created. One of the major problems previous researchers faced was the lack of proper textual data. They have been known to falsely create users to have a good-sized dataset for training. With synthetic datasets, researchers risked producing inaccurate results.

- II. A stylometric classification tool delivered as Python code that can be trained and evaluated on (I). Alongside the final model, we present results of a comparative evaluation against results from [10] [11], which demonstrates that our model achieves an accuracy of 82% in 10-fold cross-validation.
- III. Reflections and guidance on best working practices for large-scale re-identification in a resource-constrained environment. There is usually an issue with resource management for bigger datasets. In this project, we faced issues such as out of memory errors, resource reallocation errors and data imbalance errors. Even though we were dealing with the subset of the actual data as mentioned here [7], we had to get creative to solve these issues. A workaround we used was to divide the data into chunks and apply machine learning on each of that chunk. For this, algorithms that provide the *.partialfit()* method is highly desirable as it takes away the pain to keep storing the results after every chunk and then aggregating it in the end. Algorithms such as *Passive Aggressive Classifier* are really good for such operations. Apart from chunking, we also liked at paging to increase the virtual memory of the hardware we were working with. While it did help to an extent, it was certainly not enough. Hence, we had to reduce the chunk size enough to fit in the memory and use resource allocation monitors to continuously see for any potential breaks. In the case of Neural Networks, we had to come down to a batch size of as low as 8 in order to fit the data in the memory. We had a high number of pre-defined neurons and experimented with many hidden layers. As the complexity increased, the batch size had to be continuously lowered.

This project is developed using Python, applying stylometry to the underground cryptomarkets. It is designed to parse huge datasets by merging it in one source file, cleaning it and parsing it to prepare it for evaluation. Further, there will be a *model\_attempts.py* code file that will have commented algorithms that were tried and tested. However, the main logic will be written in *Model.py* file. Apart from this, we have *ExtractDetails.py* that will contain the logic to extract relevant details that were used in this project. As a sample dataset, we will also provide our complete dataset in a *CSV* file. We also make available the algorithm we used to clean the data.

The project will be made available on GitHub<sup>1</sup> post-testing and the link will be attached with this thesis. This project will interest cybersecurity professionals, lecturers, researchers, and

---

<sup>1</sup> [https://github.com/anansrivastava/Stylometry\\_UndergroundMarkets](https://github.com/anansrivastava/Stylometry_UndergroundMarkets)

employees of law enforcement agencies. It may also interest people who are looking venture in the cybersecurity field and have a keen morally positive interest in darknet markets.

The remainder of this thesis is structured as follows. In Chapter 2, we discuss the most relevant literature. We discuss the contextual background and the technical background of previous research and how it relates to our research. Chapter 3 looks at how we implemented this project, our development decisions, the dataset we worked with and the algorithms that we tried. Chapter 4 is all about the results of the algorithms we mentioned in Chapter 3. We also look at how they fared pre clean up and post-cleanup of the data. In Chapter 5, we conclude our work and discuss further steps and limitations of this research.

## 2. Literature Review

Stylometry has found many uses across different domains. Most common ones include music, literature, plagiarism, and code stylometry. In this section, we discuss each one of them.

In the world of music, stylometry was first used to analyze the work of the linguist George Zipf, who in 1949 revealed an examination of Mozart's "Bassoon Concerto in Bb Major" for instance of a framework following his very notable "Zip's Law" [12]. This law says that the occurrence of words in a book or text, for example, reduces by the power-law  $p[f] = \frac{1}{f}$  where  $p$  stands for probability and  $f$  stands for the frequency of occurrence. In simpler terms, the law states that some items will occur very often while many others, rarely. Artists now look at taking a song from a composer and breaking it down to its basic chord level, to identify if it has been copied from another song. Using a large corpus of encoded music, from the Center for Computer-Assisted Researching the Humanities at Stanford University (CCARH), music features are created. These features, known as value features, are then compared with features in the database using machine learning techniques. Different feature methods are used for different styles of songs: classical, pop, R&B etc. Stylometry is then used to connect these features in a way to make the song unique. This is done by controlling the distance between the musical notes in terms of pauses, notes and even singing parts of songs on different days. Then, a digital correction print is added on the composition to make it uniform yet unique.

In literature, stylometry has found its way into analyzing how people from different nations write. Further, it has also been used to understand the subtle differences between how men and woman write. How sentences and words are structured gives information about the writers. Each person has a unique writing style. Some use complex sentences with rich vocabulary whereas some like to form simple sentences. Even though it is often the case that a rich vocabulary can get across the emotions of the passage more vibrantly, it is not always the case. Nobel prize winner Ernst Hemingway was famous for using limited words but putting them so beautifully together that his writing has become a threshold for young writers to understand and match. Stylometry takes advantage of the fact that no two people use em-dashes, semicolons in the same way.

Another famous example of this was used by Patrick Juola, whose stylometric program helped uncover that JK Rowling was the author of a bestselling detective novel. His program called Java Graphical Authorship Attribution Program successfully picked up the similarities between the sentences and the way they were structured. He had accumulated enough evidence of resemblance between these novels and the Harry Potter books that when JK Rowling was confronted about these allegations, she did not deny them. Patrick Juola did not have to do much except create a dataset of the books and compare them to the new detective novels. He also said that the results were clear, and it took no more than 30 minutes to conclude. This research was based on the fact that how writers use function words such as article, prepositions and conjunctions, can help uniquely identify them.

## 2.1. Contextual Background

This section discusses the previous work that has been done in the field, its results and how it relevant to our study. Further, it provides a great insight into how researchers have found creative methods to deal with problems related to authorship attribution.

### 2.1.1. Underground Cryptomarkets

In this section, we give a brief overview of the origin of underground markets and the problems they pose in society. Further, we also discuss some of the previous work that has been done to understand these markets. We list the most recent and relevant studies first. We identify the research gap and build our argument based on their shortcomings. We discuss, compare, and contrast the data we have to the data that has been used by previous researchers.

Underground cryptomarkets are a part of the deep dark web that is responsible for most online illegal trading activities. Users of these markets exchange drugs, guns and some are even speculated to hire assassins. These underground markets provide a haven for these criminals as most of them take full advantage of the anonymous nature of these markets. Browsers like The Onion Routing (TOR), which provide many more functions compared to a normal browser, allow multiple rerouting to mask layers of virtual IP addresses to hide the real one. The name is a play on how an onion has multiple layers protecting its seed. With the advancement of technology and discovery of many more IP masking techniques, it has become increasingly difficult for the law enforcement agencies to track these criminals and site owners.

Underground markets first came into existence when the US government designed an anonymous software named TOR to communicate with their secret agents anonymously. When the software was made public, the expectation was that people will use to anonymously voice opinions on matters of public interest. Further, it was expected that the platform would be used by whistle-blowers to shed light on illegal activities within governments and private organizations. Instead, it soon became a platform to trade illegal drugs and the law enforcement agencies were made to look like the originators of the problem.

For law enforcement agencies (FBI, DEA), getting a grip on these large-scale, anonymous, and rapidly increasing markets has proved to be a challenge. The information gained from these markets can provide invaluable insight into how these markets operate. It helps them understand the social dynamics of the community and identify weak links. Law enforcement agencies use these insights to plan their next attack. However, these advances have been accomplished primarily through the analysis of limited structured metadata and painstaking manual work. Because of the size of the datasets and the labour intensity of the task, there are limitations to what can be accomplished using manual force.

These markets have resulted in social and financial losses in society. It is estimated [13] that a third of people from the lower functioning neighborhoods in the US are involved in these markets. Numerous studies [14] [13] have shown that people from these regions find false

comfort in the online communities as they believe that other members come from similar regions and understand the struggles of their personal lives. Drug dealers in these regions make significant money that attracts younger people. Further, lack of proper education and limited exposure to the outside world makes the younger people believe that albeit risky, this is the fastest way to earn money and improve quality of life.

A recent 2020 [7] report on how the popularity of these underground cryptomarkets grew over the last decade is astonishing. In 2019, vendors of these markets sold about \$790 million' worth of products, indicating about a 70% rise in sales from 2018. 2019 marked the first year the illegal sales crossed \$600 million. The year 2019 also saw the biggest jump in incoming transactions in cryptocurrencies, with 0.08% of transactions as compared to 0.04% in 2018. Since these are anonymous and largely illegal markets, transactions happen in cryptocurrencies like Bitcoin and Litecoin that cannot be traced. Such cryptocurrencies became very popular in the last few years and illegal trading was a major contributor to it. Figure 1 shows the trend of the usage of these cryptocurrencies.

As mentioned before, the rise of the cash flow in the underground cryptomarkets has been increasing many folds. The importance of this project stems from the fact the cybersecurity firms, law enforcement agencies like the FBI and DEA have been largely unsuccessful in regulating these markets. Further, many secret operations have been conducted by these agencies to catch and suspend some of these online marketplaces, however, research has shown that users simply migrate to other marketplaces and continue the illegal selling and purchasing. On November 6, 2014, the FBI and Europol, in collaboration with other worldwide law enforcement agencies announced that they had prevailed in a joint activity against a few shrouded administrations on the Tor darknet. The investigation, named Operation Onymous' saw the effective conclusion of 267 '.onion' pages making up 27 separate websites. Law organizations likewise made 17 captures and seized \$1 million in Bitcoin as well as different measures of money, weapons, and military arsenal. Probably the most famous website to be brought down by this operation was the Silk Road 2.0. Silk Road was notoriously famous for harbouring the darkest of the dark and deepest of the deep web buyers and vendors. The creators, Thomas White and Blake Benthall become millionaires overnight and were reportedly pulling in \$8M per month. According to the officials, the website followed in the footsteps of Silk Road 1.0 and gained traction because after it's shut down in 2013, its customers migrated to Silk Road 2.0.

Another operation called the Operation Bayonet was targeted to shut down *AlphaBay* and *Hansa* darknet marketplaces. On 20 July 2017, Europol once again joined forces with the Federal Bureau of Investigation (FBI), the US Drug Enforcement Agency (DEA) and the Dutch National Police to carry out a coordinated attack to shut down the two largest players in the dark web: *AlphaBay* and *Hansa*. This operation ranks as one of the most effective and coordinated attacks in the history of attacks against cybercrimes. *Alphabay* was launched in September 2014. Right after its launch, it grew steadily, with 14000 new users in the first 90 days.

In 2019, the German police took down the most lethal group of cybercriminals, operating in the world’s largest illegal online market, called the Wall Street Market (WSM). This marketplace had it all. They traded in cocaine, heroin, fake documents, pirated software’s, and military-grade weapons. The platform grew to a whopping 1 million customers with over 5000 vendors and 60000 sale orders. After the shutdown of *Aplhabay* and Hansa in 2017, most of their users migrated to WSM. Reports claim that the WSM was as big as Amazon and eBay in terms of capture size, however, its sole purpose of existence was to illegally traffic contrabands. It was started by 4 men, 3 German and 1 Brazilian, all of whom were arrested.

Even after the shutdown of such large marketplaces, the number of users and yearly cash flow did not decrease.

Darknet market revenue vs. Darknet market share of all cryptocurrency received by services, 2013-2019

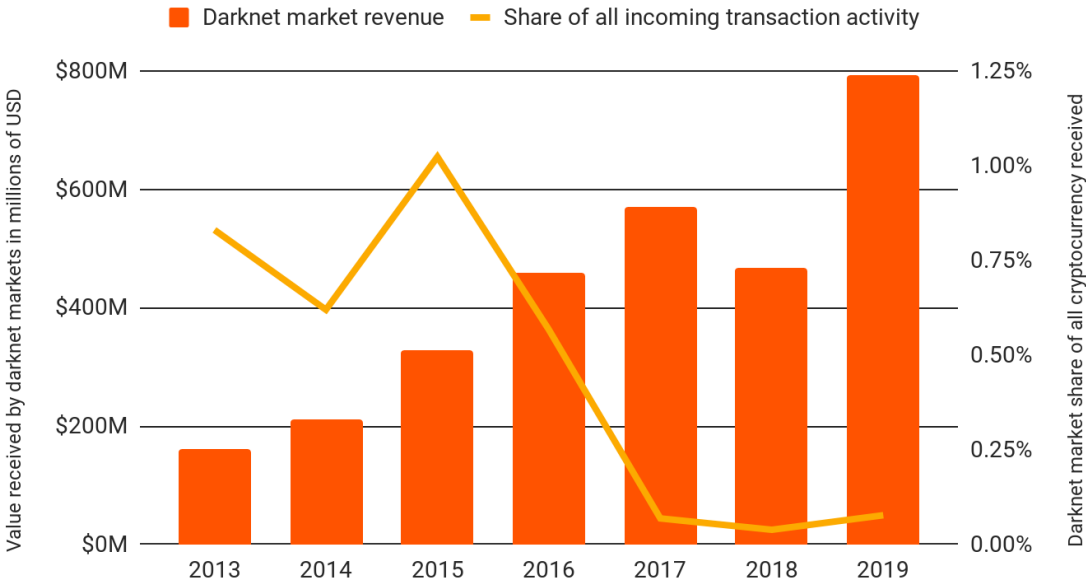


Figure 1: Darknet market revenue vs Cryptocurrency market share [7]

Further, the number of popular active darknet markets have continually grown in the last decade. From 1 active market in 2011, its popularity inspired other people to open their own marketplaces. By 2019, there 49 active markets that were fully functional. Figure 2 shows the average volume of money sent to the darknet markets over a span of 10 years. In 2018, 8 darknet markets were shut down by the law enforcement agencies, but the vendors just managed to create new marketplaces by 2019. This shows that shutting down of marketplaces had little to no effect on the problem at hand. Further, this gives insight into the operations in these markets: When a marketplace closes, there is a migration wave to other marketplaces and other marketplaces gain in popularity. Operations run as usual and the customer demand is still met.

## Number of active darknet markets vs. Average value received per darknet market, 2011-2019

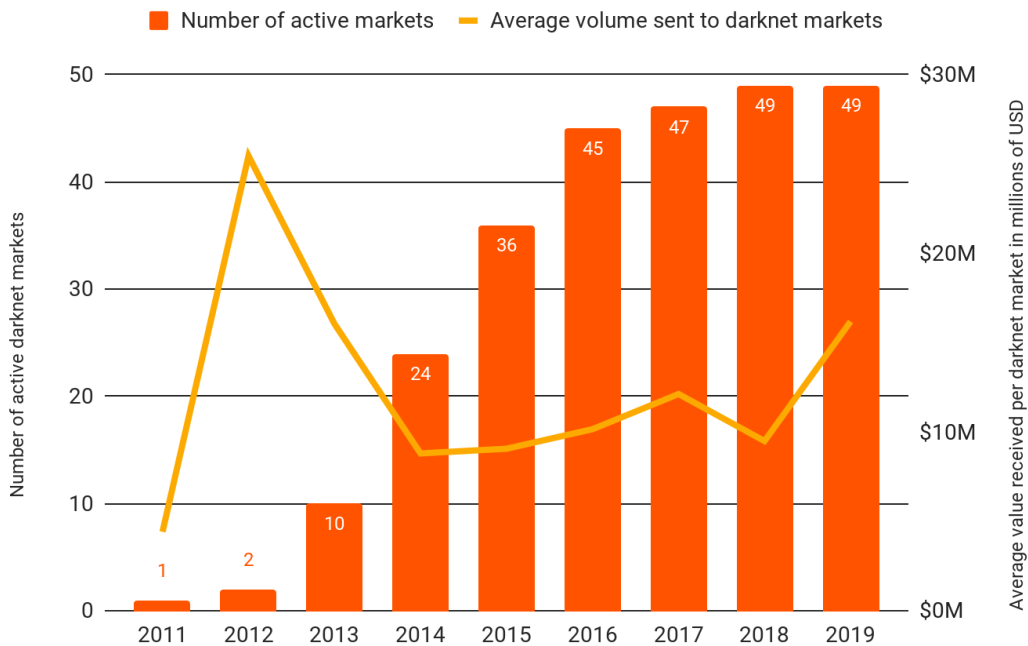


Figure 2: Active darknet markets [7]

The information above additionally asserts that the expansion in income is driven by the number of purchases rather than the size of the purchase. As evident, there was not a huge fluctuation in the purchase size in USD from 2011-2019, yet we see that the number of transactions increased from 9 million to 15 million. This suggests that either more users started purchasing from darknet markets in 2019, or that old customer simply started buying more.

### 2.2. Related Studies

History has always seen a fierce debate between anonymity and freedom of speech. A famous example that gave birth to the United States Constitution – the founding fathers wrote the highly controversial ‘federalist’ and ‘anti-federalist’ [15] paper under the pseudonyms Publis and Cato [16] [16]. The anonymous disclosure of these papers sparked nationwide unrest and people demanded the identities be revealed. Back then, anonymity gave people the power to voice their opinion more freely.

Today, anonymity on the internet is mainly used for illegal activities [3]. Underground cryptomarket users use anonymity and super anonymization softwares like *TOR* [17] to buy drugs, weapons, tanks and even hire assassins [18]. Furthermore, the situation is exacerbated by the fact that most of these users take deliberate measures (change usernames, login from different places around the globe using VPNs etc.) to protect their identities. For law enforcement agencies, it gets very difficult to track them down and capture them.



Most of the previous research on applications of stylometry in author attribution has been done on simpler datasets (blogs, tweets, Reddit comments etc.) [19] [20]. Only very few researchers have used stylometry on datasets from the underground cryptomarkets. Furthermore, of the little research that has been done on these markets, authors have mainly analyzed their metadata or products (drugs, cocaine etc.) and prices (in Bitcoins per grams) that these markets offer [21].

A study in 2000 by Rao and Rohatgi [22] did a comprehensive study on users who used different pseudonyms (usernames) on USENET newsgroups. They determined whether these users were linked to each other or not, based on their writing styles. Their strategy was to cluster 185 pseudonyms that belonged to 117 individuals. They used function words (to, for, from) and Principal Component Analysis (PCA) to cluster pseudonyms belonging to the same user. Their shortcomings were similar to that of Martijn Spitters et al. [23] Because of the lack of data (just 117 users) and a small number of unique authors, they experienced overfitting while training their model. This impacted their model performance and drastically brought down their accuracy and precision.

In another study, Afroz et al. [10] in 2014 developed a probabilistic algorithm called Doppelganger Finder. They obtained their data from L33tCrew and Carders, two popular underground cryptomarkets places. Their method found 28 pairs of users with duplicate accounts, with a precision and recall of 0.85 and 0.82, respectively. Although effective, a major drawback of their method was that it did a large amount of manual analysis to validate their results. For example, their method made use of Part-of-Speech (PoS) tagging, which is language-dependent. This required prior knowledge of the text (which is rarely the case in the real-world data) and needed installing new taggers when dealing with texts in languages other than English.

In 2015, Martijn Spitters et al. [23] analyzed these markets to find links between duplicate user accounts. They obtained their data from the Black-Market Reloaded forum, a popular underground cryptomarket place at the time. However, their dataset contained posts (chats, texts) from only 177 users. Due to insufficient data, they had to artificially split up the posts to create more data (aka oversampling). This affected their model performance. They could only obtain precision and recall of 0.45 and 0.55, respectively. These were the two major drawbacks of their research. Insufficient data and low model performance.

Soska et al. [24] looked at the use of multiple aliases (doppelganger accounts) between Silk Road 1.0, Black Market Reloaded and Sheep. They made use of the username similarity heuristic and matched aliases based on public PGP keys. From an initial list of 29,258 unique aliases, they reduced it to 9,386 vendors. These 9,386 vendors formed their ground truth data. In this project, we aim to use a similar approach. We will discuss more on this in the 'Technical Background' subsection.

One particularly intriguing study, by Koppel et al. [19] looked at using stylometry to perform author identification with a dataset of 10,000 unique authors. Their data contained blogs from

different blogging websites. This was a good study because they had enough data to build a good machine learning model. Their work, although intriguing, raised several methodological concerns, however. The authors used only 4-grams of characters as features. This meant they split up their blogs into features of 4 words. The paper does not justify why that was the case. One could assume that using hyperparameter tuning methods like *GridSearchCV* they found that for their data, 4-gram characters worked best. However, the paper does not provide any proof of it. It is unclear to what extent author identification is dependent on the context and the number of *n\_gram* words. Furthermore, character-based analysis is highly vulnerable to the topic in question. For example, a 4-gram character range might work well on a blog corpus but fail on a short messaging corpus (like on a dataset from the underground cryptomarkets). Moreover, the susceptibility of character n-gram-based features to context biases was exacerbated by the fact that the authors performed no pre-processing [11] to remove common *noise* such as signatures at the end of posts, stop words, punctuations and URLs. Therefore, their technique could not be generalized. While their results were good, their method could not be validated on a cross-context dataset.

### 2.2.1. Stylometry

Stylometry [18] is the quantitative study of literary style through computational distant reading methods. It is based on the observation that authors tend to write in a relatively consistent, recognizable, and unique ways. For example, each person has their own unique vocabulary, sometimes rich, sometimes limited. Although a larger vocabulary is usually associated with literary quality, this is not always the case. Ernest Hemingway is famous for using a surprisingly small number of different words in his writing, which did not prevent him from winning the Nobel Prize for Literature in 1954. Some people write in short sentences, while others prefer long blocks of text consisting of many clauses. No two people use semicolons, em-dashes, and other forms of punctuation in the exact same way.

In a survey of historical and current stylometric methods [19] [18], Efstathios Stamatatos points out that function words are “used in a largely unconscious manner by the authors, and they are topic-independent. For stylometric analysis, this is very advantageous, as such an unconscious pattern is likely to vary less, over an author’s corpus, than his or her general vocabulary.

Scholars have used stylometry as a tool to study a variety of cultural questions [20]. For example, a considerable amount of research has studied the differences between how men and women write [21] or are written about. Other researchers have studied ways to use stylometry to detect plagiarism in programming code [22] and music [23].

However, one of the most common applications of stylometry is in authorship attribution [24]. Given an anonymous text, it is possible to guess who wrote it by measuring certain features, like the average number of words per sentence or the propensity of the author to use “while” instead of “whilst”, and comparing the measurements with other texts written by the suspected author. As mentioned in the above section, we will use stylometry to do just that. We will apply

stylometry or linguistic analysis to identify and match authors of anonymous text in the underground cryptomarkets.

Linguistic analysis has very often been applied to many security problems before [16]: from using stylometry to identify users who create new accounts on blog sites, after they were blocked (their accounts suspended) to using topic modelling to identify illegal job postings. However, underground forums present a challenge for text analytic techniques. The messages are short and tend to mix conversations with “products” such as credit card and bank account numbers, URLs, IP addresses, etc. Furthermore, the forums are written in a multilingual 133t-speak slang that renders most natural language processing tools such as part-of-speech taggers inaccurate—this language is often intentionally difficult to parse and speak even for native human speakers and serves to weed out outsiders. As such they represent a stress test of sorts for these approaches

### 2.2.2. Author Attribution

Authorship attribution [11] is the task of identifying the author of a given text. The main concern of this task is to define an appropriate characterization of documents or texts that captures the writing style of authors. Researchers have resorted to many author attribution techniques in the past [25]. Amongst them, statistical [26] [27] and machine learning techniques have been the most commonly used methods. Even though statistical methods such as Markov chains [28], literal and grammatical statistics, and probabilistic linguistics are known for their accuracy [28], recent advancements in computing powers have made machine learning methods more favorable. As pointed out in a study by Wang et al. [29], machine learning models are more adaptive to large data and they are more tolerant of noisy data. In addition to that, they provide more options in terms of model selection (*Random Forests*, *Naïve Bayes*, *Support Vector Machines*) and data cleaning libraries (*NLTK*, *Lemmatizers*, *Stemmers*) [30] [31]. Another disadvantage of statistical models is that they are harder to implement and can fail as the data increases.

The work in 2006, by Zheng et al. [32] provided a framework to apply author attribution on forum posts. They used 5, 10, 15 and 20 sets of authors with 10 to 30 texts per author. Their feature-set comprised of lexical, structural, syntactical, and content-based features. Their method achieved an impressive accuracy of 0.976. Without structural features and content-specific features, their accuracy dropped to 0.90. One concern (and not a drawback) was their selection of data. They worked with data that had been collected over a span of 20 years. Digital writing has changed a lot since the 1990s. Hence, the accuracy of their experiment and their relevance in today’s literary sphere cannot be validated.

Building upon the work of Zheng et al. [32], Abbassi et al. [33] developed a method called ‘Writeprints’. The writeprints or unique digital signatures is a technique based on features that are more important to one author and less to another. Since these writeprints are custom-tailored to individual authors, researchers have used them to compare and determine whether

they belong to the same author or not. They tested their method on a dataset of eBay comments and observed accuracy of 91.3%. However, on a Java forum data, their accuracy came down to 52.3%. Furthermore, they mention in the paper that the drawback of this model was its performance decreased with an increase in the number of users. Also, their method was untested on a data set with multilingual 133t-speak slang.

When dealing with text classification, a real problem is that of imbalanced data. For example, some authors may have written 1000 posts and other 100. A study by Stamatatos et al. [34] aimed to study this problem, i.e. the problem of classification imbalance. They proposed two solutions to this problem. In the first solution, they created features by creating many short text samples from authors with fewer posts and less but longer text samples from authors with many posts. In the second solution, they used *oversampling* and *undersampling* along with *sequencing* to create balanced labels. They claim that both methods increased the accuracy for the minority authors with only a slight loss of accuracy for the majority authors.

## Summary

The contextual background gives an overview of underground cryptomarkets, stylometry and author attribution techniques, as well as talks about the major studies relevant to this project and their drawbacks. Some of the drawbacks were:

1. Most of the studies worked with insufficient data
2. Researchers created synthetic datasets that impacted model performance
3. Some of the studies had flawed validation techniques and hence their results could not be validated
4. Some papers did not justify the reasons behind their chosen methodologies
5. Some methods did well on smaller datasets but failed to perform with an increase in data
6. Missing data preprocessing before model training

## 2.3. Technical Background

Majority of the previous researchers have focused on analyzing the metadata like trade ratings, vendor rating, networks etc., or have made up tests to manually compare and analyze prices of the goods and products on market. Stylometry has been used to analyze internet relay channels to gain insight into trade products and their prices. One other study used stylometry to understand how they use these secure channels to expose private credentials of the common people and how this information is propagated. A separate study explored how communities link together to form social groups that can be tracked and broken down. Stylometry has also been used to study the USA and Chinese markets where it is expected that most of the drugs and other illegal products come from these countries.

Recent research has used underground cryptomarket data to upset trade activities between buyers and vendors. However, researchers have had to deal with incorrect data, small dataset,

and synthetic accounts. Thomas et al. bought twitter accounts from underground markets to identify fraudulent patterns.

### 2.3.1. Stylometric techniques

Below we talk about author attribution. We briefly describe the four author attribution techniques: string-based technique, time-profile based technique, stylometric technique and social network-based technique and motivate why we choose stylometric and time-profile based techniques for this project.

The authorship attribution methods can be split into 4 techniques: string-based technique, time-profile based technique, stylometric technique and social network-based technique.

- String-based technique: This technique looks for substrings between usernames. For example, if username 1 is, “d.prieto” and username 2 is “del\_prieto”, it is highly likely that they belong to the same person. To find such similarities between usernames can be an important feature for user linking. In this simplest of ways, this is done by string comparison and returning the number of characters matched. If the matched characters are more than a pre-defined threshold (say if the return value is greater than 6), it probably belongs to the same person. For complex datasets, researchers have been known to use the Levenshtein distance [43] and the Jaro-Wrinkler distance [44]. They calculate the ‘distance’ (similarity) between to strings of text and predict if these strings are similar or not.
- Time-profile based technique: This refers to linking users based on the time of their activities [45]. From a psychological viewpoint, people do things at the same time all their life. For example, some people eat lunch at 14:00 and some at noon. The time of their activities does not change. The time-profile technique takes advantage of that. Using this technique, we can form links between users. Like most of the people log in to their Gmail, Yahoo, Twitter and other social media accounts at the same time, similarly, users of underground markets log in to different markets at the same. We can thus look at the post timings and deduce if two accounts belong to the same person or not.

Johansson et al. [45] experimented with this idea and achieved 100% (100%) for 500 users, 89.6% (94.4%) for 2000 users, and 70.6% (79.0%) for 4000 users. The corresponding results for stylometric features were 98.2% (99.2%) for 500 users, 71.4% (78.5%) for 2000 users, and 44.6% (51.7%) for 4000 users. This supports the idea that people generally log onto their accounts at the same time. They also looked at combining stylometric features with time-based features yielding them 100% (100%) for 500 users, 93.0% (96.6%) for 2000 users, and 75.4% (82.8%) for 4000 users. However, as evident from their result, their research could not generate meaningful results with an increase in the number of distinct users.

- Stylometric technique: This technique, as has been discussed in the previous sections, looks at combining textual cues to predict whether two pieces of documents/texts have been written by the same author or not [40]. It aims to answer the question, is author B an alias of author A? This

technique, although older [19] is by far the most famous, and robust as it can find links between authors that other techniques may or may not be able to find.

- **Social-network technique:** In this analytics technique, links are formed by looking at the connections of these users. For example, if two user accounts have very similar connections, it is highly likely that they belong to the same user [45]. This technique is more used for customizing the personalized experience for users. Also, companies like Google and Amazon use this technique to show personalized products and ads to its users. Very few studies have used this technique to form links between users. It is unreliable and does not generate insightful results.

For this project, we used the time-based profile technique and the stylometric technique to form our features (date and body columns). In the underground cryptomarkets, users are careful to use pseudonyms (different usernames) to hide their identities. Hence, using the string-based technique may not work very well. Further, string-based techniques are usually used on smaller datasets as they are computationally expensive. Also, because of the anonymous nature of these markets, users do not usually form personal connections with other users. These forums are just used to talk about the product, rates [6], shipping addresses etc. Hence social networking techniques will have little to no impact on these types of datasets. Hence, stylometric technique and time-based techniques are the best two techniques to use for this project.

### 2.3.2. Oversampling & Under sampling

We often face this problem when there is a distribution imbalance like 1:100 or 1: 500 in the minority class to the majority class.

This can and does hugely affect machine learning algorithms as many tend to ignore the minority class completely. This is usually a problem as minority classes are usually the more important predictions.

To overcome this issue, we can randomly resample the training dataset. The two main ways to do this is to either delete some of the samples from the majority class, known as under sampling or duplicate data from the minority class, called oversampling.

- Random sampling is used to create balance in data for an imbalanced dataset
- Oversampling can sometimes result in heavy overfitting as it duplicates examples from the minority class
- Random under sampling is used to delete information from the majority class than can be invaluable information to a model

### **Oversampling**

This technique is used with algorithms that are partial towards a skewed distribution of data where multiple examples of the same majority class can affect the model performance. Examples of these algorithms are ones that iteratively update weights and learn coefficients, like artificial

neural networks that use stochastic gradient descent. Further, it also affects models than require decent data split support vector machines and decision trees.

There are 2 major strategies when using oversampling:

1. Minority/ Majority sampling - This works by balancing the number of data points in the minority class to the majority class. This means that if the majority class has 500 samples and the minority class has 100, this strategy would oversample the minority class so as bring it up to 500.

```
# define oversampling strategy
oversample = RandomOverSampler(sampling_strategy='minority')
```

*Figure 3: Oversampling data - Minority/Majority strategy*

2. Floating-point sampling strategy – This strategy defines the ration user wants of the minority class to the majority class. For example, 0.5 would ensure that the minority class is resampled to half the number of data points in the majority class. Example, if the majority class has 500 examples and minority class has 100 examples, it would sample up to 250 minority class examples.

```
# define oversampling strategy
oversample = RandomOverSampler(sampling_strategy==0.5)
```

*Figure 4: Oversampling data – floating point*

## Under sampling

This technique reduces the number of samples in the majority class. It is repeated until the desired level of balance is achieved. An issue with this model is that a lot of examples from the majority class is discarded. This is undesirable as loss of data can make it harder for the model to draw a line between the majority class and the minority class. This can lead to reduced classification power and lower metrics.

In this technique, the sample from the majority class is used to match the number of examples from the minority class. For example, if a dataset has 500 data points in the majority class and 100 examples in the minority class, it will resample the majority class to contain just 100 data points in the majority class. That is a loss of 400 examples.

```
# define undersample strategy
undersample = RandomUnderSampler(sampling_strategy='majority')
```

*Figure 5: Under sampling strategy*

Between the 2 strategies, although slightly more expensive, researchers and developers do tend to use the Oversampling method more frequently.

### Combining Oversampling and Under sampling

In some cases, we can achieve greater results by combining both oversampling and under sampling. For example, oversampling can be applied to the minority class to improve unfairness towards these examples, while also under sampling the majority class to reduce bias towards the majority class. Say if we have a class distribution of a dataset like 1:500, we might first use oversampling to increase the ration like 1:50 and then apply under sampling to further improve the ratio like 1:2 by deleting examples from the majority class.

```
# define oversampling strategy
over = RandomOverSampler(sampling_strategy=0.1)
# fit and apply the transform
X, y = over.fit_resample(X, y)
# define undersampling strategy
under = RandomUnderSampler(sampling_strategy=0.5)
# fit and apply the transform
X, y = under.fit_resample(X, y)
```

Figure 6: Snippet to show a combination of oversampling and under sampling

### 2.3.3. TFIDF Vectorizer

The mathematical formula of the weight of a term in a document is given by:

$W(d, t) = TF(d, t) * \log\left(\frac{N}{df(t)}\right)$  where we define  $N$  as the count of documents and  $df(t)$  as the number of documents containing the term  $t$ .

TF-IDF is a great choice as it is

- Easy to compute the similarity between 2 documents using it
- Basic metric to extract the most descriptive terms in a document
- Common words do not affect the results due to IDF (e.g., “am”, “is”, etc.)

```
def TFIDF(X_train, X_test, MAX_NB_WORDS=75000):
    vectorizer_x = TfidfVectorizer(max_features=MAX_NB_WORDS)
    X_train = vectorizer_x.fit_transform(X_train).toarray()
    X_test = vectorizer_x.transform(X_test).toarray()
    print("tf-idf with", str(np.array(X_train).shape[1]), "features")
    return (X_train, X_test)
```

Figure 7: TFIDF vectorizer



### 3. Project Implementation

In this section, we summarize the technologies needed to implement our solution. We introduce our data and look at algorithms, tools and methods that will motivate our design, implementation, and evaluation.

#### 3.1. Hardware and Software Tools

The project will be developed in Python 3.7. The development environment will be PyCharm Professional as it is one of the best IDE for Python programming. It offers exceptional support and includes plugins like auto-refactoring, Intellisense, auto code completion etc. that support fast code development. We will use GitHub as our version control repository.

The outcome of this research project is an algorithm and a predictive model that gives the probability that given two texts written by user A and user B, what is the probability that user A and user B are the same people. Techniques from stylometry and Natural Language Processing were heavily applied to collect, clean, analyze and visualize the data.

The project was developed on a Windows 10 operating system, with 16 GB RAM and 8 cores with a total processing speed of 3.8 GHz. This section will discuss the techniques used to create this author attribution tool. Multiple tests were performed under the test environment to justify the choices made. To get an overview, please refer to the flow diagram below:

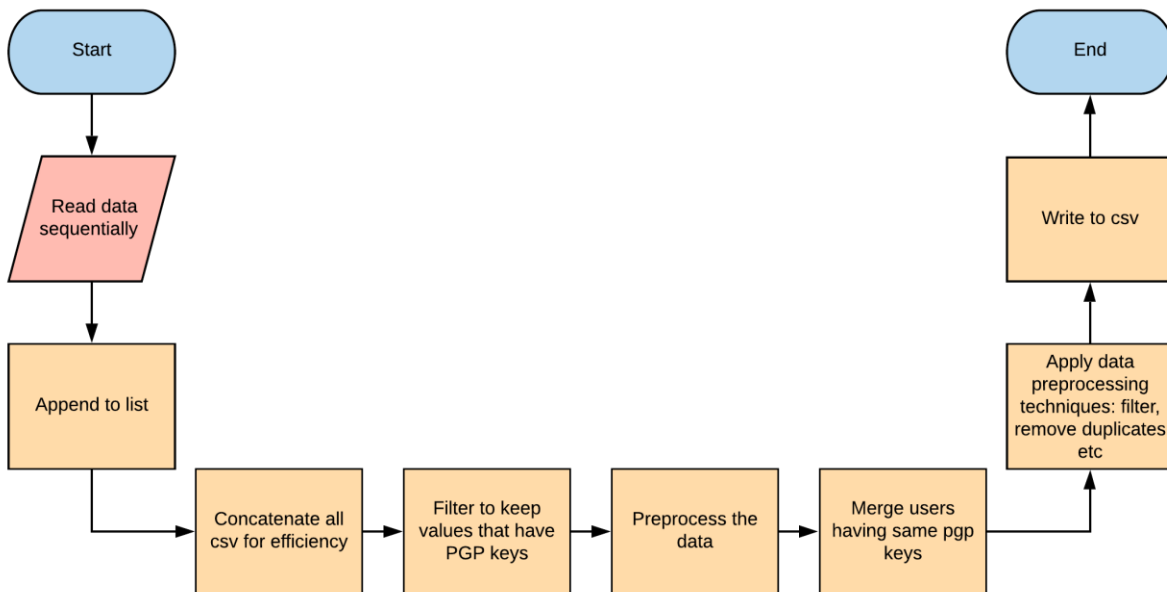


Figure 8: Parsing algorithm

### 3.2. Development Decisions

When dealing with data of a huge size, standard configured computers do not always have space or memory to allocate the data. Further, standard laptops provide a maximum of 8 cores, which is not always fast enough to keep the data in memory as well as perform other operations.

With our dataset, we realized that opening and closing 77 excel files with more than 10,000 lines of data in most of them, was very inefficient. Hence, our decision was to filter out irrelevant text or data that had very little characters or words. We eliminated users that had less than 50 words per text. Similarly, users with texts greater than 2000 words were also eliminated. Further, we kept only those users that matched for a valid PGP key. As there was no way to deal with users who did not have no data corresponding to our ground truth data. This helped us reduce the size of the data and make our program a little more efficient, while preserving more than 85% of the data. It also made sense to concatenate relevant data from all the files into one *.csv file* for efficiency.

Post concatenation of forums chats, we observed that most of the texts were from admins trying to moderate the chats. So, there were a lot of redundant messages. In datasets like these, it is not uncommon to see this behavior. Our first step was to remove these redundant messages as they would make the model biased towards one account. Further, most of the accounts had their own underground websites, personally run by them. So, there were a lot of texts with just links in them, redirecting users to personal websites of the vendors. We used regex (see Figure 3) to remove these URLs. It is common knowledge that URLs add no value to the classification power of the model. If anything, they are known to reduce it [35].

```
def remove_url(chunk):  
    chunk = chunk.str.replace('http\S+', '', case=False)  
    return chunk.replace(r'^\s*$', np.nan, regex=True) #to replace '' with  
NaN
```

Figure 9: Remove URL function

We replaced the remove URLs with *NaN* values and ultimately dropped them using *.dropna()*. Next, we observed that there were unnecessary white spaces that had to be dealt with. We used *.str.strip()* to trim white spaces and drop duplicates.

Dealing with raw textual data, written in a multilingual 133t-speak slang that renders most natural language processing tools such as part-of-speech taggers inaccurate. This language is often intentionally difficult to parse and speak even for native human speakers and serves to weed out outsiders. As such they represent a stress test of sorts for these approaches. We decided to use lemmatization and stemming both to see how they fared with each other. This is because there were spelling mistakes, capped words and slangs that rendered some of these

techniques useless. Hence, we decided to use both techniques to get the closest root word possible for the entire text.

These are some of the few steps that we used to clean our data up to a level where we could work with it to generate machine learning models. We discuss this in detail in this Chapter.

### 3.2.1. Scripting Language Selection

The main two languages for a data science project are R and Python. For this project, we chose to develop in python as we had previous experience working with it. Further, the *Pandas library* in Python provides a great way to handle datasets. Python is a complete data science language in the sense that it can be used for all the project lifecycle stages: Project Inception – Project Deployment/ Maintenance. Whereas R is mostly used for model generation. It is important to know the advantages and disadvantages of both Python and R, which is shown in Table 1:

<b>Python</b>	<b>R</b>
A more object-oriented approach	More functional approach
Suited for all the data science lifecycle stages	More used for data analysis and model creation
Extensive libraries and community support	Limited machine learning libraries
Better code readability	Difficult to understand the syntax

Table 1: Comparison between Python and R for data science

For this project, we decided to code in Python because of its flexibility, support and ease of use. Further, it is relatively easier to install and 3<sup>rd</sup> party libraries that may be needed for people working on this project in the future.

### 3.2.2. Environment

The initial data exploration was done in Jupyter notebooks. This is because Jupyter notebooks give more control on running parts of the program. We used Jupyter notebooks to clean the data, look for outliers, analyze graphs, compress the data to keep texts within a certain range etc. Libraries such as *Seaborn* provide great control over different ways to look at charts. We plotted character length to compute average characters per text. We plotted word length to compute average words per text. This helped us get a sense of the magnitude of the data. Further, charts such as box plot helped us get the percentile range, outliers and other information needed when dealing with textual data.

Post data clean up and data analysis, we switched to PyCharm workbook. PyCharm Professional is one of the best IDE for Python programming. It offers exceptional support and includes plugins like auto-refactoring, Intellisense, auto code completion etc. that support fast code development. For version control repository, we used GitHub. Another reason to use PyCharm is that it offers inbuilt GitHub push/pull controls. So the user never has to leave the environment for any operation.

### 3.2.3. Dataset

The data for this project (forums chats from underground cryptomarkets) was obtained from a publicly available collection released by an independent researcher. It is hosted at [9]. We worked with a subset of this collection. Our data looks at user posts from 77 different underground cryptomarkets. About 1.2 million rows of data. There are 2 file types. One looks at the registration of these users (columns: *community*, *user\_id*, *title*, *first\_seen* and *public\_key*) and the other deals with their conversations on these forums (columns: *community*, *user\_id*, *date*, *subject*, *category*, *body* and *quotes*). As mentioned in the Contextual Background section where we talk about a study by Soska et al. [24], we matched the PGP keys of different users to form links between them. This confirmed if two user accounts belonged to the same user or not. This formed part of our ground truth data.

To get the number of people with the fake/linked accounts, we compared the '*public\_key*' column that contains the PGP keys of users. Using *pandas.merge()*, we merged the data based on matches in the '*public\_key*' column.

```
pd.merge(excelLinks.reset_index(), excel2.reset_index(), on='public_key',
how='inner', suffixes=('_left',
'_right')).dropna().query("index_left!=index_right").drop(
columns=["index_left", "index_right"])
```

Figure 10: Snippet to merge multiple CSVs

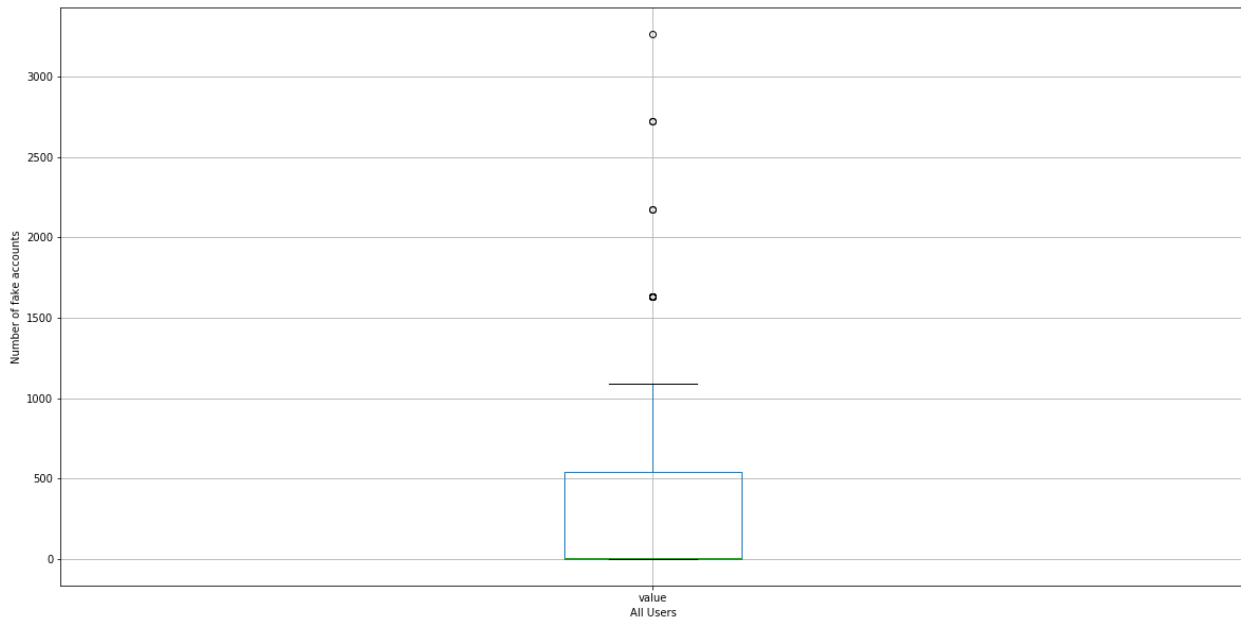


Figure 11: Box plot for fake accounts of all users

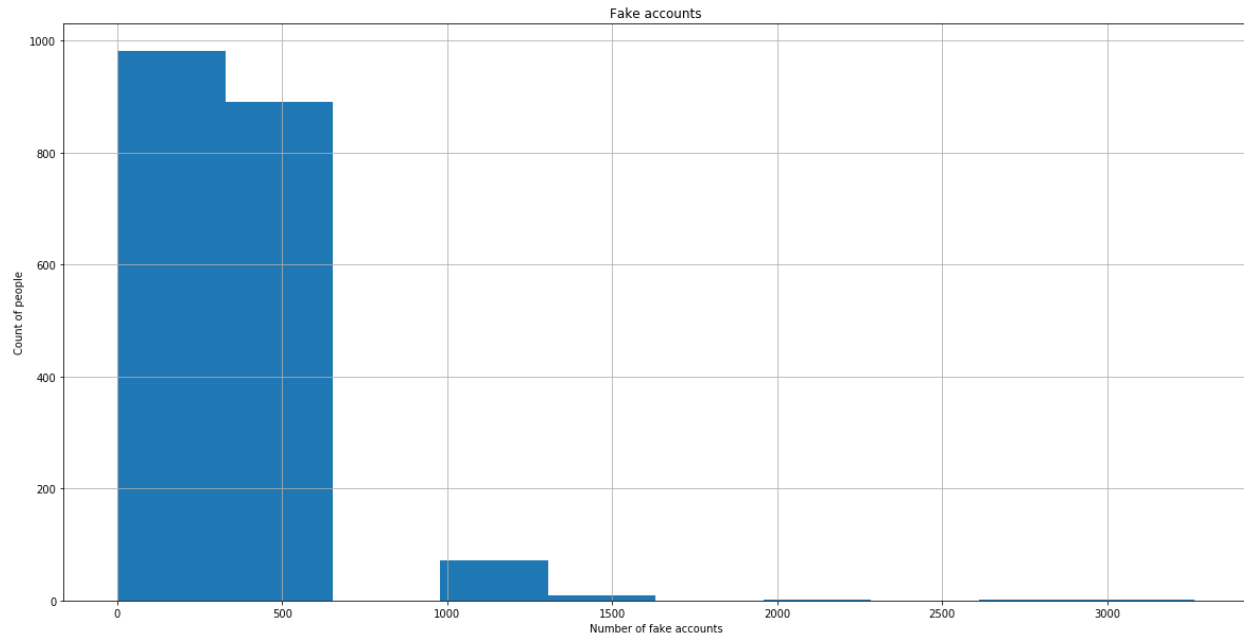


Figure 12 People with fake accounts

The chart above describes how users have many accounts on different forums using either the same username or different. Since it was impossible to plot data for everyone, we show the count of the people and the number of fake accounts they have. For example, about 35 people had more than 1100 fake accounts.

We also looked at the number of posts per author. This gave us a sense of how imbalanced our dataset was. Some users like “*CaptainWhiteBeard*” had close to 4500 posts whereas others had less than 50. We used the method `nltk.counter()` to count the number of entries

```
author = list(data['user_id'].values)
x = nltk.Counter(author)
print(x)
```

Figure 13: Library to count the number of posts per author

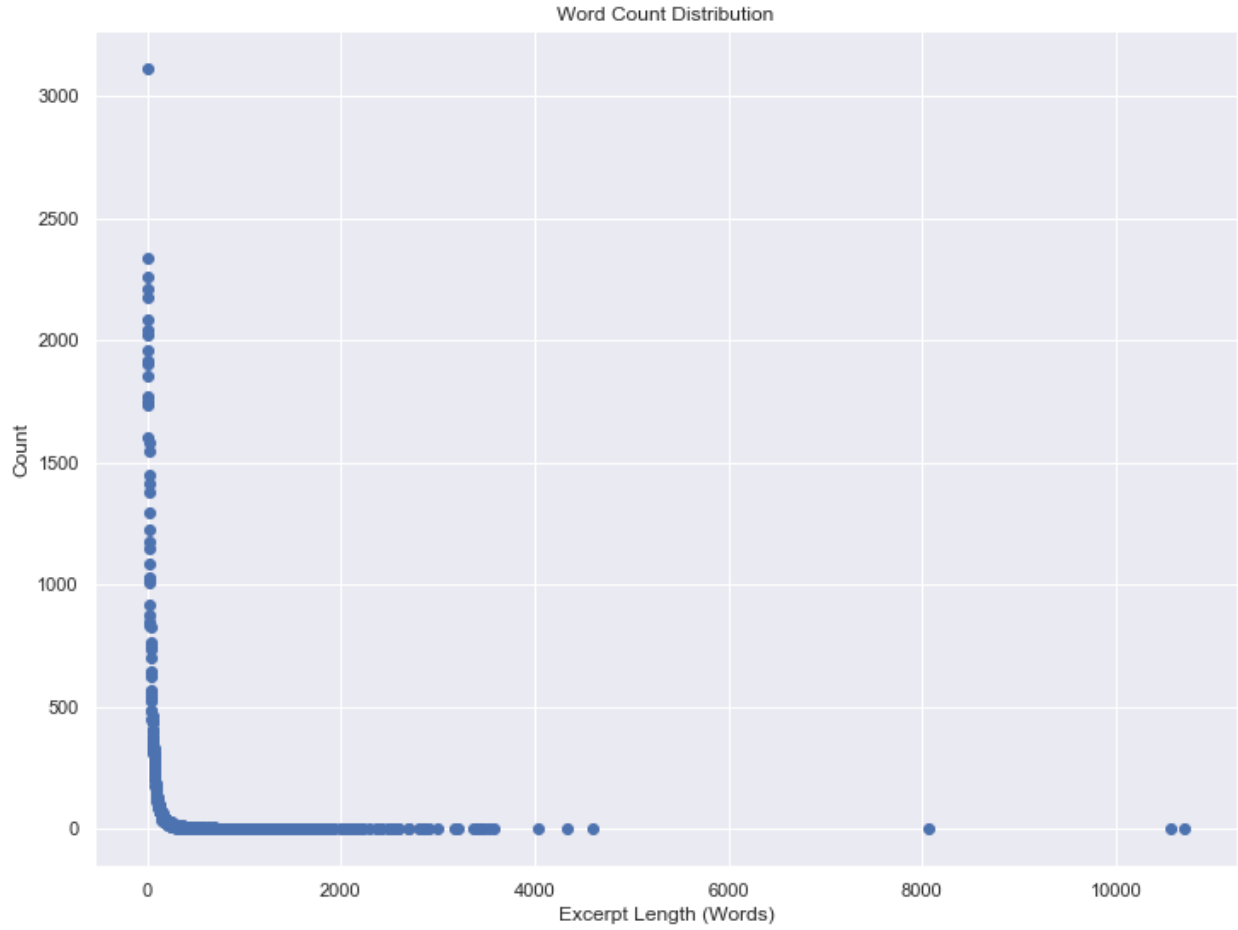


Figure 14: Word count distribution

<b>Word count statistics</b>
<b>Min: 1</b>
<b>Max: 10713</b>
<b>Mean: 51.919092163067546</b>
<b>Median 24.0</b>
<b>1st percentile 1.0</b>
<b>95th percentile 174.0</b>
<b>99th percentile 440.3000000000029</b>
<b>99.5th Percentile 629.0</b>
<b>99.9th Percentile 1547.9500000001572</b>

Table 2: Word count statistics

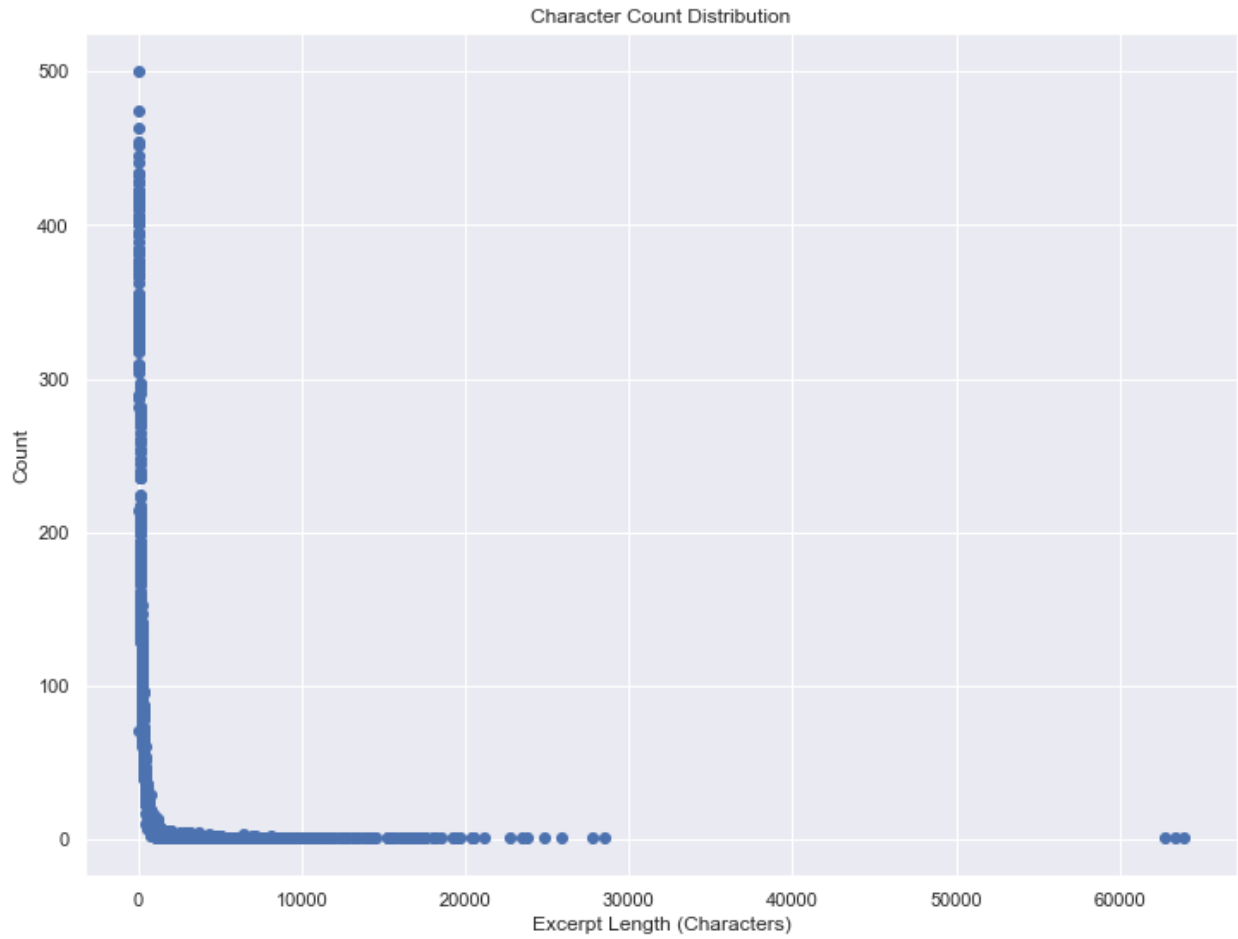


Figure 15: Character count distribution

<b>Character count statistics</b>
<b>Min: 1</b>
<b>Max: 63997</b>
<b>Mean: 292.7049064124427</b>
<b>Median 118.0</b>
<b>1st percentile 1.0</b>
<b>95th percentile 1014.0</b>
<b>99th percentile 2929.5899999999965</b>
<b>99.5th Percentile 4131.2949999999998</b>

**99.9th Percentile 10808.659000000014**

Table 3: Character count statistics

The data clearly shows that it would have been pointless to keep some of these users as they had far too many or far too few contributions towards these forums. Hence, we set a limit to only deal with data from users that had more than

$2000 \geq \text{string length} \geq 50$  and

$18000 \geq \text{character length} \geq 5$ .

We arrived at these numbers by assessing the word count and character count as shown in the above graphs and tables.

```
mask_words = (text.str.split().str.len() > 50) & (text.str.split().str.len() < 2000)
text = text.loc[mask_words]
label = label.loc[mask_words]
mask_chars = (text.str.len() > 5) & (text.str.len() < 18000)
text = text.loc[mask_chars]
```

Figure 16: Snippet keeping posts between the desired range

#### 3.2.4. Efficiency

Considering the size of the dataset, the two major problems we had anticipated was resource allocation and memory error. Opening and reading of 77 CSV files with more than 10000 lines of data in every one of them would have taken too much processing power, making it slow, time consuming and inefficient. We were also worried about permanent damages like a memory leak. Further, the author did experience random shut down of applications like Google Chrome when trying to process the entire dataset as input to a machine learning model at once. After careful consideration of options, the best (and fastest) solution was to merge all the data from all the CSV files to create one master CSV file. Further, this file was processed to remove duplicates, missing values and only keep values that had concrete PGP keys to form an association between different users across different forums.

The second biggest challenge we faced was when training our model and deciding which classifier to go with. We tried 14 different algorithms with different parameters to detect which one worked best for our data. We will discuss this in detail in the Algorithm Selection section. To continuously train on such a big dataset, we took the below-defined steps.

- Paging – a concept where you use the hard disk space of your machine to convert it into virtual RAM for more memory, and



- Chunking – Pandas library allows handling huge datasets by breaking them and dividing them into chunks of set size. This allows performing model training on smaller chunks that reduce training time, thus reducing the space-time complexity.
- Change column type – Depending on the environment and the language of choice, *pandas* library automatically creates int32, int64, float32, float64 columns for numerical ones. Below are the different subtypes and their memory usage.

SUBTYPES	MEMORY	RANGE
INT8 / UINT8	1 byte	-128/127 or 0/255
BOOL	1 byte	true or false
FLOAT16 / INT16 / UINT16	2 bytes	-32768/32767 or 0/65535
FLOAT32 / INT32 / UINT32	4 bytes	-2147483648 and 2147483647
FLOAT64 / INT64 / UINT64	8 bytes	

Table 4: Memory usage of different data types

- Down casting column type – Pandas stores objects as categorical columns. This type of storage demands high memory capacity since it creates a pointer list that points to the memory address of each value in the column. For columns that have less than 50% unique values (lower cardinality), this can be achieved by forcing *pandas* to use a virtual mapping table, where unique values are mapped via integers and not pointers.

```
# downcasting a float column
df['col1'] = pd.to_numeric(df['col1'], downcast='float')
# downcasting an integer column
df['col2'] = pd.to_numeric(df['col2'], downcast='unsigned')
```

Figure 17: Snippet to downcast column types

### 3.2.5. Clean Code

We followed best coding practices throughout the project. Maintaining clean and clear code was the secret sauce behind the success of this project. When dealing with a production-level project, having multiple files, the code must be well structured, easy to understand and modify. Further, it was essential to making sure that any updates and modifications to one part of the code did not affect other parts. For example, modifications to the dataset did not affect the actual data. Where intuitive, different sections of the code were coded on separate files, tested, and verified before integrating it with the main code.

In addition to that, we used GitHub as our version control repository to have the flexibility to go back a commit if the new changes introduced major bugs or broke the flow of the code.

### 3.3. Clean Up Strategies

For our dataset, we started by getting a description of the dataset. *Pandas* library provides a method called `.describe()` that can give us information like size of the dataset, the number of columns and rows, how many missing values are present, amongst other things like mean and median of the entire dataset.

Since we had more than a million rows of data points, texts with less than 5 characters were removed. These rows were just one or two words that would have added little to no value to the classification power of the model. Only columns that were relevant to our dataset were kept [*market name, date timestamp, text*]. Below are the strategies that we used to clean up our data:

- Remove unwanted characters
- Tokenization
- Removing stop words
- Lemmatization/Stemming

We discuss these techniques in the below sub-sections.

#### 3.3.1. Removing unwanted characters

This is an important step in data cleaning. In our dataset, we domain address to these underground markets sites which was nothing but unwanted noise. So, it is important to get rid of all the HTML tags, XML sources etc. Figure 3 shows how this can be achieved. Along with that, syntactic analysis can help check the grammar of the text by comparing it against the formal rules of grammar. Other things like punctuation characters, non-alphabets, and other characters that do not hold up against the rules of grammar must be removed to improve the quality of the dataset. To do this, we make use of regular expressions. They are a powerful way to filter our unwanted texts and symbols.

```
def remove_html(text):  
    soup = BeautifulSoup(text, 'lxml')  
    html_free = soup.get_text()  
    return html_free
```

Figure 18: Snippet to remove HTML tags

However, making a regular expression that works efficiently is a challenge. They can be hard to comprehend and form. Hence, people usually do not prefer to use them or use them only to filter out simpler things. Having said that, there are online sites such as *regexr.com* that can aid in developing meaningful regular expressions, tailored to ones needs.

This is, however, not always the case. In some cases, retaining the full stops and exclamation marks etc. becomes important. For example, in the case of classifying whether tweets from a tweets dataset are angry, neutral or happy, we can perform sentiment analysis to figure it out. A tweet that reads, “Protestors being beaten and jailed” vs “Protestors being beaten!!!! And jailed..!!!?” convey two different meanings. The first way suggests that while the user is sad, he/she is just trying to put that out in the open for the general public to be aware of what is happening. The second tweet suggests that the user is disgusted and is very angry with what is happening with the protestors. So same words could mean different sentiments. The only way we were able to differentiate here was by analyzing the over usage of punctuations. It gives a sense of that *extra* feeling of disgust the user is trying to convey.

Text-based emoticons also play an important role when performing sentiment analysis. These come in handy when you want to detect sarcasm. Emoticons such as “:), :(, -\_- , :D, xD” help determine the meanings of tweets or other texts.

### 3.3.2. Tokenization

This is the technique that helps perform both semantic as well as syntactic analysis. Tokenization is a way to help break paragraphs into sentences and sentences into words.

```
import nltk
def tokenize_sentence(text):
    return nltk.tokenize.sent_tokenize(text)
```

Figure 19: Tokenize paragraphs into sentences

```
import nltk
def tokenize_sentence(text):
    tokenized_words = [nltk.word_tokenize(i) for i in text]
    return tokenized_words
```

Figure 20: Tokenize sentences into words

In addition to tokenization, it may be a good idea to convert everything into lowercase. Again, this will largely depend on the type of system we are building. If the intensities of the emotions do not matter, then it should be fine to convert everything into lowercase. However, if we want to consider the sentiments behind texts, something written in uppercase might mean different than something written in lowercase. For example, “Let us go get that deal” and “LET’S GO GET THAT DEAL” have same words but the latter shows the enthusiasm behind it.

### 3.3.3. Removing Stop Words

These are words that are used very often but do not offer much semantic meaning. They are called stop words because they have been known to not offer much value. Words like ‘the, a, of’ etc. are some of the examples.

There are multiple ways to remove stop words. We discuss the two most used methods here. The first method is to count the individual stop word occurrences and remove them based on the threshold provided. The other way is to determine the dictionary/list of *stopwords* which can then be removed from the text.

```
def remove_stopwords(text):  
    words = nltk.word_tokenize(text)  
    tokenized_words = [w for w in words if w not in  
stopwords.words('english')]  
    return tokenized_words
```

Figure 21: Removing stop words using NLTK

### 3.3.4. Lemmatization/Stemming

Both techniques are used to find the root word of a word. It is good to use them both, but it is important to know the difference. Stemming is the more aggressive of the two and so people generally do not prefer it. It cuts off the endings of words based on common ones. Often, the new word loses its semantic meaning as the 'too' much of a root word.

Lemmatization is more suited for analysis. It gives the true root word. It works by mapping common words into one base. It involves vocabulary and some morphological analysis.

```
lemmatizer = nltk.stem.WordNetLemmatizer()  
w_tokenizer = nltk.tokenize.WhitespaceTokenizer()  
def lemmatize_text(text):  
    return [lemmatizer.stem(w) for w in w_tokenizer.tokenize(text)]
```

Figure 22: Lemmatization of words

Once again, these processes should be used only when necessary as affixes of text can contain important information. For example, 'cooler' and 'coolest' have the same root word but carry different semantic meanings. So, based on your application, one will need to decide whether to use these techniques or not and if yes, which one to use.

All these methods were used in our models.

## 3.4. Algorithm Selection

Algorithm selection is the most important part when working on a data science project. However, trying out multiple algorithms is costly. It is not only time consuming but also expensive in terms of memory. Hence, using techniques like cross-validation help us get a sense of how well the algorithm is performing.

Cross-validation is a technique used to resample data and evaluate machine learning models on a chunk of the complete dataset. It works by defining  $k$ , an inbuilt parameter that defines the

number of groups the data must be split into. Hence, the procedure is often called k-fold cross-validation. This technique is primarily used to evaluate how well does a machine learning model perform on unseen data. This method is popular because it generally alleviates the bias in a model, that can be seen in other models, such as the train test split. A general procedure may look like this:

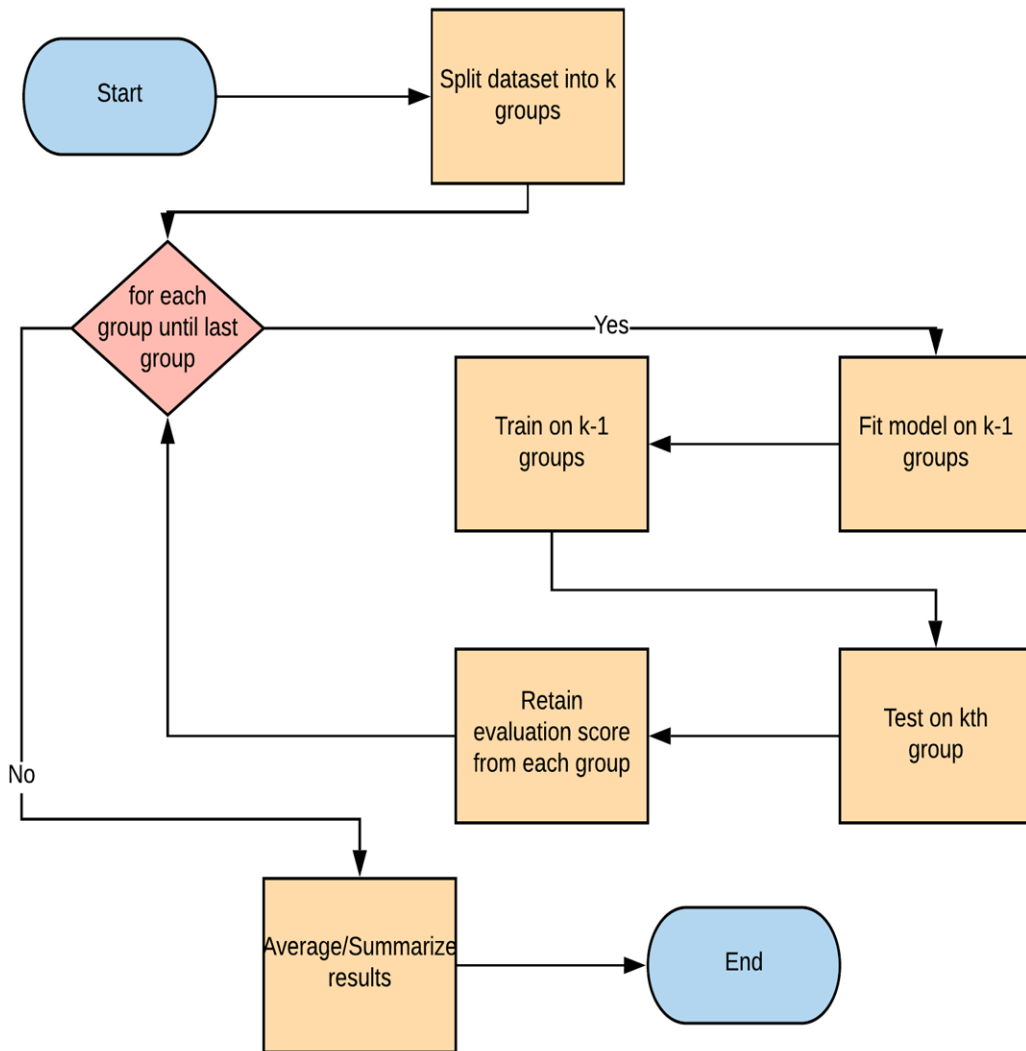


Figure 23: Cross-validation steps

Each model is fed the textual data i.e. the chats that were scraped from these forums. Some of the messages included sharing of card details and bank details, addresses to deliver. Buyers usually only shop from trusted vendors and enquire about the price of new products such as illegal drugs and weapons. As mentioned before, we determine which two (or more) users are the same based on the PGP links.

This information was stored in a separate '*PGPHardlinks*' CSV file, which was used for training models as they formed our true examples. This CSV file contained only users that were in the range of our users with minimum and maximum number of words and characters (see chart above). This was necessary to do as the data was highly imbalanced. For example, "*CaptainWhiteBeard*" had close to 4500 posts whereas some had less than 50.

From all the registration files, we first extracted all users (*community, user\_id, first\_seen, public\_key*) that had a PGP key associated with them. Then, for every user extracted, the PGP key of that user was compared with the PGP keys of all the other users. If two (or more) PGP keys matched, a positive association was formed between those 2 (or more) users. It confirmed that they are the same people under a different (or same) username, operating on different (or same) websites. For example, a user had two pseudonyms on the same marketplace (*Abraxus Forum*): '*FreeTradeInc*' and '*FreeTrade*'. Here we can see that the same person has different usernames on the same website. Another example, a user with username '*ContinusTOR*' on *Abraxus* was operating as '*Yaoming – UKevolution*' on *Alphabay* and '*Yaoming – uk*' on *The Hub*. Notice how this person is using different pseudonyms on different websites but is the same person.

After the associations were formed and confirmed (using PGP keys) between different users who were actually the same people, chats were extracted from the other CSV file in the dataset that contained all the details from the forums of these markets for these users. In other words, for each user that we had positive confirmation of having multiple accounts, every text and message they wrote was extracted, from every marketplace in our dataset (77 marketplaces). The final CSV, named '*ExtractedData.csv*' contained the columns: *user\_id, date, body, community*. This file formed the basis for our training and testing data. Chats from this CSV file were cleaned, visualized, and fed as input to the different machine learning models.

As for our train and test examples, we chose our dataset to have only those users who had matched for PGP links and were in the CSV file mentioned above. Data from these users were split in train and test using K-fold and used for making predictions. As it had almost all users in the CSV anyway, we had a rich dataset and the loss of users with no PGP links was minimal.

The scikit-learn class provides a method called *KFold()*. It takes *k* as its arguments and whether the data needs to be shuffled. It is always advisable to shuffle data before splitting as there are fewer chances of bias. An example of this is as follows:

```
#sample snippet
kfold = KFold(10, True)
```

Figure 24: *KFold* snippet

The data can then be split using the *.split()* function. This function is called repeatedly, and it will return each group of the train and test sets.

```
for train, test in kfold.split(data):
    print('train: %s, test: %s' % (train, test))
```

Figure 25: Snippet to divide data into test and train splits

We used *Kfold()* in our project to evaluate model performance.

There are 4 stages to selecting an algorithm that works best for your dataset. With the training data, we try numerous algorithms to get a good model prediction. On the validation dataset, we tweak the parameters of the algorithm to make it biased towards our data. Post this, we test and evaluate model performance based on the metric scores on the test data. Below are some of the initial algorithms we tried for our dataset. In some of the methods, we used the following text processor

```
def text_process(tex):
    # 1. Removal of Punctuation Marks
    nopunct = [char for char in tex if char not in string.punctuation]
    nopunct = ''.join(nopunct)
    # 2. Lemmatisation
    a = ''
    for i in range(len(nopunct.split())):
        b = lemmatiser.lemmatize(nopunct.split()[i], pos="v")
        a = a + b + ' '
    # 3. Removal of Stopwords
    return [word for word in a.split() if word.lower() not
            in stopwords.words('english')]
```

Figure 26: Data cleaning function used in the project

We explain below the settings and libraries we used when trying out different models for our dataset. Since most of the models have some common setting, we define them here to avoid repetition.

1. Label Encoder – Machine learning models cannot deal with textual data. Textual data must be converted into numerical data beforehand. And to do this, we make use of Label Encoder. Label Encoder can be found in the *sklearn* preprocessing library and is used to fit and transform text data with the new encoded data.
2. Standard Scaler – Standard scalers are very helpful when dealing with outliers. It will help transform the entire dataset or the columns on which it is used such that the distribution has a mean value of 0 and a standard deviation of 1. The standard score of a sample is calculated as  $z = (x - u) / s$ .
3. *Train\_Test\_Split* – This is a function that splits the data into two parts – training data and testing data. This is a very useful function by *sklearn* and it takes away the effort of manually portioning the data. Further, the test size can be defined as a parameter that

takes a numerical value. You can also shuffle the data before portioning to reduce chances of bias.

4. TF-IDF Vectorizer - Used to reflect on how important a word is to the document. It increases proportionally to the number of times the word appears in a document but is offset by the frequency of the word to consider that some words usually appear more times. TF-IDF vectorizer transforms text data into feature vectors that are used as model inputs.
5. Pandas Chunking – Pandas provides a way to read large CSVs breaking them down into smaller chunks. This is usually done to reduce memory usage. When dealing with large datasets, it is common to face out of memory/memory allocation error problems. A chunk is defined as a part of the dataset. The size of it depends on the amount of RAM that we have. Below is the method to the following chunking:
  - a. Read data in chunks by specifying the chunk size – `pd.read_csv(chunksize = x)`
  - b. Process the chunk for data cleaning and model estimation
  - c. Save the results from this chunk
  - d. Repeat steps a to c until the whole dataset is handled
  - e. Combine the results
6. Part of Speech Tagging (POS) – POS are used to create parse trees that contain information about whether a word is a noun, named entity, verb etc. They are also used in lemmatizing words. It considers not just the word but also the context in which it is being used. However, it is generally difficult, and the task is not straightforward, as the same word could have a different meaning in different contexts. POS has 4 different techniques
  - a. Lexical Based Methods - This method assigns the POS tag the most frequently occurring with a word in the training corpus.
  - b. Rule-Based Methods – This method uses regex and rules to assign POS tags. For examples, words ending with *'ed'* or *'ing'* can be classified as verbs. A combination of Rule-based methods and Lexical based methods can be used to tag unseen words that are present in the testing data but not in the training data
  - c. Probabilistic Methods – This method uses Conditional Random Fields (CRFs) and Hidden Markov Models (HMM) to predict the probability of a particular tag sequence occurring.
  - d. Deep Learning Methods – RNNs and ANNs can also be used for POS tagging.



7. Encoding Data – Tensor flow preprocessing data class allows a text corpus to convert into a sequence of integers or vectors where the coefficient for each token may be binary.

```
def encode_data(docs):  
    tokenizer = text.Tokenizer(num_words=1000)  
    tokenizer.fit_on_texts(docs)  
    tokenized = tokenizer.texts_to_sequences(docs)  
    docs = sequence.pad_sequences(tokenized)  
    return docs
```

Figure 27: Snippet to encode textual data

8. Feature Creation –

```
nn_list = []  
vb_list = []  
for sentence in docs:  
    nns = 0  
    vbs = 0  
    elements = sentence.split('_')  
    for item in elements:  
        if item == 'NOUN':  
            nns += 1  
        if item == 'VERB':  
            vbs += 1  
    nn_list.append(nns)  
    vb_list.append(vbs)  
nn_list = np.array(nn_list).reshape(-1, 1)  
vb_list = np.array(vb_list).reshape(-1, 1)
```

Figure 28: Feature creation

9. Keras Classifier – The Keras classifier taken in an argument 'build\_fn' which used to call the model name. It requires you to define the function call and return the created model. We use it in our code to call the 'build\_model' function that compiles are model and returns the final model. Moreover, we use parameters like epochs, early stopping and batch size to fine-tune our model.

```
estimator = KerasClassifier(build_fn=build_model(feature_nodes,  
feature_nodes, 1000), epochs=3, batch_size=512,  
verbose=1, callbacks=[EarlyStopping(patience=3,  
monitor='accuracy'])],  
validation_data=(X_test, y_test))
```

Figure 29: Snippet to show usage of Keras Wrapper

10. Cross-Validation – Cross-validation is a technique used to resample data and evaluate machine learning models on a chunk of the complete dataset. It works by defining  $k$ , an inbuilt parameter that defines the number of groups the data must be split into. Hence, the procedure is often called  $k$ -fold cross-validation. This technique is primarily used to evaluate how well does a machine learning model perform on unseen data. This method is popular because it generally alleviates the bias in a model, that can be seen in other models, such as the train test split.
11. Early stopping – One of the bigger challenges in machine learning is wondering how long to train a model before it starts to learn the noise from the data. If gone unchecked, it can result in a problem called exploding gradients. *Early stopping* is a *Keras* call back function that stops model training when the accuracy of the model with each epoch starts reducing. It has a parameter called '*patience*' that can be set to an integer number that keeps the training running to see if the model improves the performance.
12. Pretrained glove embedding – *GloVe* is an unsupervised learning algorithm used to obtain vector representation for words. *GloVe* stands for "Global Vectors for Word Representation". It is a somewhat popular embedding technique based on factorizing a matrix of word co-occurrence statistics. Our approach using Pretrained glove embedding was as follows:
  - a. Create word index sequence by converting word samples. A word index is nothing but an integer id of the word.
  - b. Construct the "embedding matrix" which will contain at index  $i$  the embedding vector for the word of index  $i$  in our word index.
  - c. Input the embedding matrix created in step b into the *Keras* embedding layer.
  - d. Construct the neural network on top of it. Our network was a 1D neural network with a *softmax* output activation.
13. Pad-Sequences – This function transforms samples of sequences into a list of integers. It takes the form of a 2D NumPy array of shape  $(num\_samples, num\_timesteps)$ .  $num\_timesteps$  is either the *maxlen* argument if provided, or the length of the longest sequence in the list. *num\_timesteps* is used as a threshold to measure sequences. Sequences are truncated or expanded by adding 0 paddings until they are *num\_timesteps* long. It is used to ensure all the sequences in a list have the same length.
14. Embedding matrix – An embedding matrix is a list of all the words from a document and their corresponding word index/embedding term. For example, a dictionary may contain terms like {I: "1", like: "2", Chinese: "3", Italian: "4"} and sentence maybe "I like Chinese". In this case, it will imply that **like (2) > I(1) or Italian (4) > Chinese (3)**. This poses a problem. If we input this in our machine learning model, it will learn unwanted behaviour and may be biased towards one class than other. Hence, we need embedding matrices.

- a. I – [1,0,0,0]
- b. like – [0,1,0,0]
- c. Chinese – [0,0,1,0]
- d. Italian – [0,0,0,1]

Now, we can write the sentence as  $A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ . This eliminates the problem as there is no strict comparison between Chinese and Italian. In this above equation, A is known as the embedding matrix.

15. *ReduceLROnPlateau* – This callback function is used to reduce or increase the learning rate when the model hits a plateau. It is usually used when the model stops improving. Just like Early Stopping, it is a callback function that is used as a parameter while training the model.

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
                             patience=5, min_lr=0.001)
model.fit(X_train, Y_train, callbacks=[reduce_lr])
```

Figure 30: Snippet to show how to adjust the learning rate

### 3.4.1. MultinomialNB

The NB stands for Naïve Bayes classifier. It is used for classification when the dataset comprises of discrete features. Features such as the word count of text classification. Usually, they need to be converted into a numerical equivalent such as integers. However, floats may also be used as they work relatively well with either. Naïve Bayes usually finds its importance in NLP problems as they can be used to predict the tags of the text. In our case, tags are *user\_ids* and text is the chat from the forums. They calculate the probability of all tags for given text input and output the tag with the highest probability. Hence, it is usually inadvisable to use MultinomialNB for extremely large dataset.

Model settings and libraries used

- *CountVectorizer(analyzer = text\_process)*
- *train\_test\_split(X, y\_label, test\_size = 0.3, random\_state = 42, shuffle = True)*
- *MultinomialNB()*

### 3.4.2. Logistic Regression

Logistic regression measures the relationship between labels (dependent variables) and features (independent features) using its underlying logistic regression function called sigmoid. It is a machine learning algorithm used to fit data onto a single line and divides the data into two or more halves so that the data can be linearly classified.

Model settings and libraries used:

- *LabelEncoder()*
- *preprocessing.StandardScaler()*
- *train\_test\_split(test\_size = 0.4, random\_state = 42, shuffle = True)*
- *LogisticRegression(max\_iter = 7000, solver = 'saga')*

### 3.4.3. LinearSVC

It is similar to an SVC model but with a linear kernel. This model scales better and is known to have a large number of parameter options. It scales better to large data and has support for both dense and sparse input.

Model settings and libraries used:

- *TfidfVectorizer(ngram\_range = (1,2), analyzer = text\_process)*
- *train\_test\_split(test\_size = 0.3, random\_state = 1337)*
- *LinearSVC()*

### 3.4.4. Passive Aggressive Classifier

In recent times, using this classifier method has become extremely popular as it allows partial training of the data. It has a method called partial fit that allows training of large dataset in batches. It is an incremental learning algorithm. The core concept is that the classifier adjusts its weight vector for each misclassified training sample it receives, trying to get it correct.

Model settings and libraries used:

- Pandas chunking - *pd.read\_csv("Data.csv", chunksize = 5000)*
- *numpy.unique(incremental\_target)*
- *TfidfVectorizer(ngram\_range = (1,2))*
- *train\_test\_split(vectors, incremental\_target, test\_size = 0.3, random\_state = 3)*

- `passive_aggressive = PassiveAggressiveClassifier()`
- `passive_aggressive.partial_fit(X_train, y_train, classes)`

### 3.4.5. Neural Networks model 1

Model settings and libraries used:

```
def build_model(embed_size, max_length, vocab_size):
    def build_model():
        model = Sequential()
        model.add(Embedding(vocab_size, embed_size, input_length=max_length))
        model.add(Conv1D(embed_size, 7, activation='relu', padding='same'))
        model.add(MaxPooling1D(2))
        model.add(GlobalMaxPool1D())
        model.add(Dropout(0.1))
        model.add(Dense(50, activation="relu"))
        model.add(Dropout(0.1))
        model.add(Dense(output_nodes, activation="sigmoid"))
        model.compile(loss='binary_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])
        plot_model(model, to_file='model_plot.png', show_shapes=True,
                   show_layer_names=True)
        return model
    return build_model
```

Figure 31: Neural network structure

```
def pos_tag(docs):
    nlp = spacy.load('en_core_web_sm')
    tagged_sentences = []
    for item in docs:
        combined = []
        tagged = nlp(item)
        for token in tagged:
            combined.append(' '.join([token.lemma_, token.pos_]))
        combined_string = ' '.join(combined)
        tagged_sentences.append(combined_string)
    return tagged_sentences
```

Figure 32: POS tagging function

- *Part of Speech Tagging* – both noun and verbs
- *Feature Creation*
- *Encoding data*

- *Label Encoder*
- *loss function – binary\_crossentropy*
- *KerasClassifier(build\_fn = build\_model(130,130,20000),epochs = 3,batch\_size = 10000,verbose = 1)*
- *KFold(n\_splits = 10,shuffle = True,random\_state = 128)*
- *cross\_val\_score(estimator = estimator,X = train\_data,y = train\_labels,cv = folds)*

### 3.4.6. Neural Networks Model 2

Model settings and libraries used:

```
def build_model(embed_size, max_length, vocab_size):
    model = Sequential()
    model.add(Embedding(vocab_size, embed_size, input_length=max_length))
    model.add(Conv1D(embed_size, 7, activation='relu', padding='same'))
    model.add(MaxPooling1D(2))
    model.add(GlobalMaxPool1D())
    model.add(Dropout(0.5))
    model.add(Dense(150, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(output_nodes, activation="sigmoid"))
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    plot_model(model, to_file='model_plot.png', show_shapes=True,
               show_layer_names=True)
    return model
```

Figure 33: Neural network structure

- *Part of Speech Tagging*
- *Feature Creation*
- *Encoding data*
- *Label Encoder*
- *loss function – categorical\_crossentropy*
- *build\_model(130,130,75000)*
- *EarlyStopping(patience = 2,mode = 'min',monitor = 'val\_loss',verbose = 1)*
- *estimator.fit(X\_train,y\_train,verbose = 1,batch\_size = 512,validation\_split = 0.2,callbacks = [early\_stopping\_monitor],nb\_epoch = 5)*

### 3.4.7. Neural Networks Model 3

Model settings and libraries used:

```
def Build_Model_DNN_Text(shape, nClasses, dropout=0.5):  
    """  
    buildModel_DNN_Tex(shape, nClasses, dropout)  
    Build Deep neural networks Model for text classification  
    Shape is input feature space  
    nClasses is number of classes  
    """  
    model = Sequential()  
    node = 1024 # number of nodes  
    nLayers = 8 # number of hidden layer  
  
    model.add(Dense(node, input_dim=shape, activation='relu'))  
    model.add(Dropout(dropout))  
    for i in range(0, nLayers):  
        model.add(Dense(node, input_dim=node, activation='relu'))  
        model.add(Dropout(dropout))  
    model.add(Dense(nClasses, activation='softmax'))  
  
    model.compile(loss='sparse_categorical_crossentropy',  
                  optimizer='adam',  
                  metrics=['accuracy'])  
  
    return model
```

Figure 34: Neural network structure

- *Part of Speech Tagging*
- *Feature Creation*
- *Encoding data*
- *Label Encoder*
- *loss function – sparse\_categorical\_crossentropy*
- *EarlyStopping(patience = 2, monitor = 'val\_loss')*
- *model\_DNN.fit(X\_train, y\_train, validation\_data = (X\_test, y\_test), epochs = 5, batch\_size = 64, verbose = 2, callbacks = [early\_stopping\_monitor])*

### 3.4.8. Neural Networks Model 4

Model settings and libraries used:

```
def build_model(embed_size, max_length, vocab_size):
    model = Sequential()
    model.add(Embedding(vocab_size, embed_size, input_length=max_length))
    model.add(Conv1D(embed_size, 7, activation='relu', padding='same'))
    model.add(MaxPooling1D(2))
    model.add(GlobalMaxPool1D())
    model.add(Dropout(0.5))
    model.add(Dense(150, activation="relu"))
    model.add(Dropout(0.5))
    model.add(Dense(output_nodes, activation="sigmoid"))
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    # plot_model(model, to_file='model_plot.png', show_shapes=True,
    show_layer_names=True)
    return model
```

Figure 35: Neural network structure

- *Part of Speech Tagging*
- *Feature Creation*
- *Encoding data*
- *Label Encoder*
- *EarlyStopping(patience = 2, monitor = 'val\_loss')*
- *estimator.fit(X\_train, y\_train, verbose = 1, batch\_size = 16, validation\_split = 0.2, callbacks = [early\_stopping\_monitor], epochs = 200)*



### 3.4.9. Neural Network Model 5

Model settings and libraries used:

```
def Build_Model_DNN_Text(shape, nClasses, dropout=0.5):
    """
    buildModel_DNN_Tex(shape, nClasses, dropout)
    Build Deep neural networks Model for text classification
    Shape is input feature space
    nClasses is number of classes
    """
    model = Sequential()
    node = 1024 # number of nodes
    nLayers = 5 # number of hidden layer

    model.add(Dense(node, input_dim=shape, activation='relu'))
    model.add(Dropout(dropout))
    for i in range(0, nLayers):
        model.add(Dense(node, input_dim=node, activation='relu'))
        model.add(Dropout(dropout))
    model.add(Dense(nClasses, activation='softmax'))

    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])

    return model
```

Figure 36: Neural network structure

- *Part of Speech Tagging*
- *Feature Creation*
- *Encoding data*
- *Label Encoder*
- *EarlyStopping(patience = 2, monitor = 'val\_loss')*
- *model\_DNN.fit(X\_train, y\_train, validation\_data = (X\_test, y\_test), epochs = 5, batch\_size = 64, verbose = 2, callbacks = [early\_stopping\_monitor])*

### 3.4.10. Pretrained Glove Embedding

Model settings and libraries used:

```
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = Conv1D(EMBEDDING_DIM, 5, activation='tanh')(embedded_sequences)
x = MaxPooling1D(2)(x)
x = Dropout(0.2)(x)
x = Conv1D(EMBEDDING_DIM, 5, activation='tanh')(x)
x = MaxPooling1D(2)(x)
x = Dropout(0.2)(x)
x = Conv1D(EMBEDDING_DIM, 5, activation='tanh')(x)
x = GlobalMaxPooling1D()(x)
x = Dense(EMBEDDING_DIM, activation='tanh')(x)
x = Dropout(0.2)(x)
preds = Dense(output_nodes, activation='softmax')(x)
model = Model(sequence_input, preds)
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Figure 37: Neural network structure

- glove.6B.100d.txt
- `LabelEncoder()`
- `Tokenizer(num_words = MAX_NUM_WORDS)`
- `pad_sequences(sequences, maxlen = MAX_SEQUENCE_LENGTH)`
- `num_words = min(MAX_NUM_WORDS, len(word_index) + 1)`  
`embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))`
- # load pre – trained word embeddings into an Embedding layer  
# note that we set trainable = False so as to keep the embeddings fixed  
`embedding_layer = Embedding(num_words,`  
`EMBEDDING_DIM,`  
`embeddings_initializer = Constant(embedding_matrix),`  
`input_length = MAX_SEQUENCE_LENGTH,`  
`trainable = False)`
- `EarlyStopping(patience = 3)`

- `ReduceLRonPlateau(monitor = 'val_loss', factor = 0.1, patience = 2, min_delta = 1E - 7)`
- `model.fit(x_train, y_train, batch_size = 128, epochs = 30, validation_data = (x_val, y_val), verbose = 1, callbacks = [es, rlrp])`

The above-mentioned algorithms are just a few examples of the different algorithms we tried with different combinations and hyperparameters. Overall, we tested 14 algorithms. Since we observed the best results with `LinearSVC()` and `PassiveAgressiveClassifier()`, we decided to explore more on these methods using hyperparameter tuning with `GridSearchCV()`. We discuss this in detail in the Results section.

## 4. Results

This section will discuss the functionality of the author attribution tool and how we evaluated the performance of the model. To guarantee a consistent outcome, we tested the model performance on the unseen data for users that were matched to have valid PGP links. In the below table, we discuss the results of every algorithm tested and described in Section 3.4.

### 4.1. Pre clean up

The initial dataset encompassed a lot of noise in the form of garbage values, missing data, invalid characters etc. Pre data cleanup, the metrics looked very weak. This is because when dealing with textual data, it is important to vectorize the words into numerical values. So the more garbage values we have, the more outliers. Please see the below table:

#### 4.1.1. Classification Report

ALGORITHM	ACCURACY	PRECISION	RECALL	F1-SCORE	TIME
<b>MULTINOMIAL NB</b>					
NO	0.28	0.21	0.08	0.09	1320.26
PARAMETER OPTIMIZATION					
PARAMETER OPTIMIZATION	0.30	0.24	0.11	0.12	1318.48
<b>LOGISTIC REGRESSION</b>					
NO	0.12	0.11	0.05	0.09	1080.37
PARAMETER OPTIMIZATION					
PARAMETER OPTIMIZATION	0.16	0.11	0.10	0.11	1088.29
<b>LINEAR SVC</b>					
NO	0.52	0.49	0.50	0.48	462
PARAMETER OPTIMIZATION					
PARAMETER OPTIMIZATION	0.69	0.65	0.62	0.66	1074
<b>PASSIVE AGGRESSIVE CLASSIFIER</b>					
NO	0.62	0.59	0.60	0.61	16
PARAMETER OPTIMIZATION					
PARAMETER OPTIMIZATION	0.69	0.68	0.66	0.61	36

<b>NEURAL NETWORK 1</b>					
NO PARAMETER OPTIMIZATION	0.68	0.60	0.63	0.65	7200
PARAMETER OPTIMIZATION	0.63	0.58	0.59	0.59	7650
<b>NEURAL NETWORK 2</b>					
NO PARAMETER OPTIMIZATION	0.43	0.45	0.45	0.42	8629
PARAMETER OPTIMIZATION	0.47	0.46	0.47	0.48	8531
<b>NEURAL NETWORK 3</b>					
NO PARAMETER OPTIMIZATION	0.32	0.24	0.30	0.27	7150
PARAMETER OPTIMIZATION	0.28	0.15	0.19	0.22	7741
<b>NEURAL NETWORK 4</b>					
NO PARAMETER OPTIMIZATION	0.62	0.64	0.63	0.65	5368
PARAMETER OPTIMIZATION	0.63	0.60	0.60	0.61	6225
<b>NEURAL NETWORK 5</b>					
NO PARAMETER OPTIMIZATION	0.11	0.15	0.14	0.12	8540
PARAMETER OPTIMIZATION	0.12	0.13	0.15	0.11	9649
<b>PRETRAINED GLOVE EMBEDDING</b>					
NO PARAMETER OPTIMIZATION	0.28	0.24	0.27	0.26	1320
PARAMETER OPTIMIZATION	0.29	0.27	0.25	0.26	1427

Table 5: Different algorithm results

## 4.2. Post clean up

Our neural network algorithms gave deceptively good results. We observed extreme overfitting as they learned noise from the candidate sets. Since we observed the best results from Linear SVC and Passive-Aggressive Classifier, we decided to work ahead with them. The metrics drastically improved post cleaning up the data. Our cleanup strategies are mentioned in Section 3.3.

### 4.2.1. Results – Post cleanup

ALGORITHM	ACCURACY	PRECISION	RECALL	F1-SCORE	TIME
<b>LINEAR SVC</b>					
NO PARAMETER OPTIMIZATION	0.76	0.77	0.76	0.74	680
PARAMETER OPTIMIZATION	0.82	0.80	0.82	0.79	1243
<b>PASSIVE AGGRESSIVE CLASSIFIER</b>					
NO PARAMETER OPTIMIZATION	0.77	0.75	0.78	0.72	63
PARAMETER OPTIMIZATION	0.79	0.72	0.78	0.76	89

Table 6: Linear SVC results - post-cleanup

As evident, LinearSVC gave us the best results out of all the algorithms that we tried. We mark that as our final selected algorithm for this research.

## 4.3. Summary



This project made use of the 14 algorithms to figure out which model worked best for our dataset. With highly imbalanced data, we observed that neural networks observed extreme overfitting and learned noise as well. Actual performance on the unseen dataset was rather unsatisfactory.

LinearSVC gave us the based results for us. It takes a parameter  $C$ , which is the regularization parameter. Its strength is inversely proportional to  $C$ . We tried a range of 1 to 10 for this parameter and observed the performance improved as we increased  $C$ . Moreover, compared to other algorithms it took considerably less amount of time to train the model.

#### 4.4. Critical Evaluation

We aimed to answer that given two accounts, what is the probability that they belong to the same person, based on the conversations received from both the accounts. For this, we achieved 82% accuracy, 80% precision, 82% recall and 79% F1-score using LinearSVC. The research was done based on previous research in this field and identifying their weak points like using synthetic datasets, fake usernames, different twitter or Reddit datasets and not actual datasets from the underground market.

We also aimed to develop a strong parsing algorithm to clean, visualize and parse a dataset of text written in a multilingual 133t-speak slang. This is different and a tougher ball game as existing techniques may not always be enough to clean a dataset of this nature.

<i>Objectives</i>	<i>Result</i>
<i>Apply stylometry to underground markets</i>	
<i>Create a robust algorithm to parse data of this sort</i>	

*Table 7: Objectives met*

##### 4.4.1. Uniqueness

Our approach of limiting users within a range of words and characters, alongside keeping only those users that had a PGP link associated with them, so as not to create synthetic data is what drove this research to success. Further, it helped to have enough real-world data to train a machine learning model to classify more than 1000 users correctly.

As we have mentioned in section 2.0, each user has a distinct writing style. SVCs work very well when there are distinct classes. Further, they work well with unstructured or semi-structured data such as text and trees. The sigmoid kernel of the SVCs does a great job classifying separate classes. Other researchers have also concluded that they have had good results from using SVMs and SVCs.

Since we have worked with real datasets and none of the data was artificially created, the results will hold for all type of reidentification, as long as the dataset is in English. However, on a very small dataset with textual data of less than 50 words per user, the algorithm will fail. This is because, in our research, we discarded such users and kept only users who had data of words between 50 and 2000 words. Moreover, the results could probably be improved a notch if we

used an inbuilt spelling checker to check and correct spellings. These factors would affect the results of the research.

The major strengths of this research are that it can be further explored by law enforcement agencies like Cyber Security Teams, FBI, and DEA to form links between multiple users. Further, it can also be used to explore other concepts like author obfuscation that, if coupled with the methodologies of this research, will strengthen then results at the very least. One advice to people using this tool will be to continue to explore this space using deep learning methods. While they did not work very well for our case, deep learning is the future and people working with this tool are encouraged to try and implement deep learning sequential methods to get even better results.

***Summary:***

This project met both of its predefined objectives.



## 5. Conclusion

This research aimed to find doppelganger accounts in the underground cryptomarkets. This research was done to contribute towards making the cyberspace safer by dealing with cybercriminals who leak personal information of people around the world or trade illegal products. This project was exciting to work on as the challenge of working with NLP techniques is already a hard one but coupled with it being implemented on a dataset from underground cryptomarkets, which is still a niche space, is fascinating. Having read some of the chats of these people on these forums, one realizes how grave and dire some of the thoughts of these people are and what threat it poses on our society. It gives an insight into the thought process of these people and a world which is best left untouched. This project aimed to uncover some of its workings and identify these cybercriminals based on their writing style.

In section 1, we introduced and motivated the idea behind this project and briefly discuss the objectives, value-added and deliverables. In section 2, we look at the remarkable work that has been done by previous researchers. We also look at the technical background where we discuss the methods previous researchers have used and what we use for this project. Section 3 talks about the mechanics of this project, our development decisions and algorithms that were tested. In the results sections, we discuss our findings and explain why we think our results hold for this project. LinearSVC works well with textual data as has been corroborated by previous researchers. In this section, we discuss our take on this project, possible extensions to this project for future work and limitations.

### 5.1. Scope Extension

While we achieved our objectives, there is always room for improvement. As mentioned in this section, we can look at the concept of Author Obfuscation to make our solution more robust and fail-proof. We discuss more on that below.

#### 5.1.1. Author Obfuscation.

Author obfuscation [49] [50] is a technique used to evade author identification. This technique is either performed manually, are computer-assisted, or are entirely automated [51]. This technique could be studied to determine how it plays a role in the stylometric analysis of texts [35] [49]. This would, however, increase the complexity of the project due to the 3 techniques being highly advanced and hard to comprehend [52]. It would also need ground truth data to verify whether there was a deliberate attempt by authors to obfuscate their writing styles. Obtaining that data for our dataset is a difficult task. In a fully mature system, we would have this data already and look at ways to incorporate this into our project.

### 5.1.2. Factors that Influence User Decisions

A famous example of user migration: when the FBI and Europol jointly conducted the operation, 'Operation Onymous' [4] and shut down Silk Road 2.0. This caused unrest in other big underground cryptomarkets. As a result, most of the users decided to migrate [4] [34] to other smaller underground cryptomarkets. If possible, we will further investigate factors like these (shutting down of marketplaces) to better understand the key elements that drive user decisions.

### 5.1.3. Incorporating datasets from languages other than English

As for the improvement areas, this research only aimed at language spoken in English and treated words and symbols from other languages as outliers. Libraries like *py-translate* can be used to detect and convert words from other languages and be considered. This might further improve the metrics and incorporate a larger dataset from non-English speaking countries.

## 5.2. Limitations

In our dataset, it made sense to only keep the words from the English language as most of the words from our dataset were in English. Hence, the very little foreign language words were treated as outliers. Consequently, any word of a foreign language was classified as an invalid word and did not contribute to the training of the model.

In addition to that, we assumed that the data was original and genuine and not already obfuscated as there was no way ground truth data to prove otherwise.

## 5.3. Future Works

This project will be open for people to use and download via GitHub. This project can be expanded to incorporate stylometry for different languages. Further, this project can also be used to identify plagiarism and author obfuscation.

## References

- [1] A. Greenberg, "Feds Seize Silk Road 2 in Major Dark Web Drug Bust," 11 June 2014. [Online]. Available: <https://www.wired.com/2014/11/feds-seize-silk-road-2/>.
- [2] M. Yip, N. Shadbolt and C. Webber, "Why forums? An Empirical Analysis into the Facilitating Factors of Carding Forums," *Proceedings of the 3rd Annual ACM Web Science Conference, WebSci 2013*, 2013.
- [3] G. Weimann, "Terrorist Migration to the Dark Web," in *Perspectives on Terrorism Vol 10*, Terrorism Research Initiative, 2016, pp. 40-44.
- [4] D. Décary-Hétu and L. Giommoni, "Do police crackdowns disrupt drug cryptomarkets? A longitudinal analysis of the effects of Operation Onymous.," *Crime Law Soc Change* 67, vol. 67, no. 1, pp. 55-75, 2017.
- [5] T. Babor, *Drug policy and the public good*, Oxford University Press, 2010.
- [6] J. Franklin, A. Perrig, V. Paxson and S. Savage, "An inquiry into the nature and causes of the wealth of internet miscreants," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 375-388, 2007.
- [7] C. Team, "Darknet Market Activity Higher Than Ever in 2019 Despite Closures. How Does Law Enforcement Respond?," 28 January 2020. [Online]. Available: <https://blog.chainalysis.com/reports/darknet-markets-cryptocurrency-2019>.
- [8] J. Mullin, "Silk Road 2.0, infiltrated from the start, sold \$8M per month in drugs," Arstechnica, 2014.
- [9] G. Branwen, N. Christin and D. DécaryHétu, "Darknet Market Archives , 2011–2015," 12 07 2015. [Online]. Available: <https://www.gwern.net/DNM-archives>.
- [10] S. Afroz, A. C. Islam, A. Stolerman, R. Greenstadt and D. McCoy, "Doppelgänger Finder: Taking Stylometry to the Underground," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 212-226.
- [11] M. Koppel, J. Schler and S. Argamon, "Authorship Attribution in the Wild," *Language Resources and Evaluation*, vol. 45, no. March 2011, pp. 83-94, 2011.
- [12] P. ST, "Zipf's word frequency law in natural language: a critical review and future directions," *Psychonomic Bulletin & Review*, vol. 5, pp. 1112-1130, 2014.

- [13] J. J and P. G, "Black Youths and Illegal Drugs," *Journal of Black Studies*, vol. 32, p. 422–438, 2002.
- [14] C. Windsor, B. E and D. E, "Dimensions of Oppression in the Lives of Impoverished Black Women Who Use Drugs," *Journal of Black Studies*, vol. 41, pp. 21-39, 2010.
- [15] A. Furtwangler, *The Authority of Publius: A Reading of the Federalist Papers.*, Ithaca, New York: Cornell University Press, 1984.
- [16] *McIntyre v. Ohio Elections Commission; 514 U.S. 334; Supreme Court of the United States*, 1995.
- [17] R. Dingleline, N. Mathewson and P. Syverson, "Tor: The Second-Generation Onion Router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, San Diego, CA. USA, USENIX Association, 2004, pp. 21-22.
- [18] S. Martijn, V. Stefan and v. S. Mark, "Towards a Comprehensive Insight into the Thematic Organization of the Tor Hidden Services," in *2014 IEEE Joint Intelligence and Security Informatics Conference*, The Hague, 2014, pp. 220-223.
- [19] M. Koppel, J. Schler, S. Argamon and E. Messeri, "Authorship Attribution with Thousands of Candidate Authors," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Seattle, Washington, USA, Association for Computing Machinery, 2006, pp. 659-660.
- [20] M. Almishari, D. Kaafar, E. Oguz and G. Tsudik, "Stylometric Linkability of Tweets," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, Scottsdale, Arizona, USA, Association for Computing Machinery, 2014, pp. 205-208.
- [21] K. Peretti, "Data Breaches: What the Underground World of Carding Reveals," *25 Santa Clara Computer & High Tech*, pp. 375-376, 2008.
- [22] J. R. Rao and P. Rohatgi, "Can Pseudonymity Really Guarantee Privacy?," in *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9*, Denver, Colorado, USA, USENIX Association, 2000, pp. 7-8.
- [23] M. Spitters, F. Klaver, G. Koot and M. v. Staalduinen, "Authorship Analysis on Dark Marketplace Forums," *European Intelligence and Security Informatics Conference*, pp. 1-8, 2015.
- [24] S. Kyle and C. Nicolas, "Measuring the Longitudinal Evolution of the Online Anonymous Marketplace Ecosystem," in *Proceedings of the 24th USENIX Conference on Security Symposium*, Washington, D.C., USENIX Association, 2015, pp. 33-48.

- [25] T. C. Mendenhall, "A mechanical solution of a literary problem," *Popular Science Monthly*, vol. 60, no. 2, pp. 97-105, 1901.
- [26] B. Michael and G. Rachel, "Practical Attacks Against Authorship Recognition Techniques," in *Innovative Applications of Artificial Intelligence*, 2009.
- [27] J. F. Burrows, "Word-Patterns and Story-Shapes: The Statistical Analysis of Narrative Style," *Literary and Linguistic Computing*, vol. 2, pp. 61-70, 1987.
- [28] D. Khmelev and F. Tweedie, "Using Markov Chains for Identification of Writers," *Literary and Linguistic Computing*, vol. 16, 2002.
- [29] R. Y. Wang, V. C. Storey and C. P. Firth, "A framework for analysis of data quality research.," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, pp. 623-640, 1995.
- [30] S. V and A. R, "Evaluating Preprocessing Techniques in Text Categorization," in *International Journal of Computer Science and Application Issue*, 2010.
- [31] A. Kadhim, "An Evaluation of Preprocessing Techniques for Text Classification," in *International Journal of Computer Science and Information Security*, 2018.
- [32] Z. Rong, L. Jiexun, C. Hsinchun and H. Zan, "A framework for authorship identification of online messages: Writing-style features and classification techniques," *Journal of the American Society for Information Science and Technology*, vol. 57, no. 3, pp. 378-393, 2005.
- [33] A. Abbasi and H.-c. Chen, "Writeprints: A Stylometric Approach to Identity-level Identification and Similarity Detection in Cyberspace," *ACM Transactions on Information Systems*, vol. 26, pp. 1-29, 2008.
- [34] E. Stamatatos, "Author identification: Using text sampling to handle the class imbalance problem," *Information Processing & Management*, vol. 44, no. 2, pp. 790-799, 2007.
- [35] S. Gharatkar, A. Ingle, T. Naik and A. Save, "Review preprocessing using data cleaning and stemming technique," *International Conference on Innovations in Information, Embedded and Communication Systems*, pp. 1-4, 2017.