İSTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS

DETECTING SOCIAL BOTS USING TWITTER DATA

Graduation Project Final Report

150180043: Yusuf Alptigin Gün
150170034: Yavuz Kanber

Department: Faculty of Computer and Informatics Engineering
Division: Computer Engineering

Advisor: Prof. Dr. Mehmet KESKİNÖZ

June 2022

İSTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS

DETECTING SOCIAL BOTS USING TWITTER DATA

Graduation Project Final Report

150180043: Yusuf Alptigin Gün
150170034: Yavuz Kanber

Department: Faculty of Computer and Informatics Engineering
Division: Computer Engineering

Advisor: Prof. Dr. Mehmet KESKİNÖZ

June 2022

Statement of Authenticity

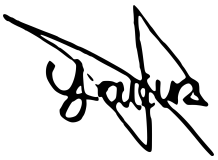
I/we hereby declare that in this study

1. all the content influenced from external references are cited clearly and in detail,
2. and all the remaining sections, especially the theoretical studies and implemented software/hardware that constitute the fundamental essence of this study is originated by my/our individual authenticity.

İstanbul, June 2022

Yusuf Alptigin Gün

Yavuz Kanber



DETECTING SOCIAL BOTS USING TWITTER DATA

(SUMMARY)

Today, online social networks are an important part of our lives. Twitter is one of the most used online social networking sites. While Twitter has accounts that are controlled by humans, it also has accounts that are controlled by computer algorithms. Accounts that are controlled by computer algorithms are called bot accounts. Bot accounts can be both beneficial or malicious, depending on the purpose of the creator. Sharing fake news, phishing, political campaigns, and increasing an accounts follower count are some of the examples of the use of malicious accounts. Our goal is to design a system that people can use when they want to check if an account is a bot.

Our system will use machine learning algorithms to detect social bots. We will test different algorithms and the best resulting algorithms are what we will use in our system to detect bots. While training our algorithms, the goal will be to use data sets that were used in past research and get better results. In the data sets we will be using, users' Twitter account data and their tweet data will be available. By using the features that are already in the data set and extracting some of our own, we will combine them to train our models.

Our system to detect social bots on Twitter consists of 3 steps. Our first step in the system is designing a machine learning model with the best results we can achieve. This machine learning model will be the key part of detecting social bots on Twitter. In the next step, we will create a website. This website will be used to gather the profile and tweet data of the users that are being asked to check if they are a bot. In the last step, the gathered data by the website will be sent to the machine learning platform where the results from that same platform will be shown to the user on the website.

By extracting a different set of features from previous works, our model will have higher accuracy in detecting social bots. We will include sensitivity and specificity analysis of our models. To make our models more generalizable, we will use different data sets. We will also provide a fine user experience for our users with our unique user interface that is easy to understand.

While creating our system, we will deal with non-functional and functional requirements. We will deal with non-functional requirements such as reliability, usability, performance, maintainability, scalability, portability, supportability, and security while also dealing with functional requirements. These functional requirements will include things like the users' ability to detect social bots, the system having a simple user interface, the system's ability to respond in adequate time, etc.

Our system needs laptops for software development, Python, HTML, a development environment for these coding languages, and data sets to train our models. We divided our project into 4 different parts. These parts are namely initiation, design, development, and testing & publishing. In the initiation part of our project, we do things like system design. These types of things are required to be done at the start of the project. In the design part, we have things we need to learn and prepare such as algorithms needed for our system, website development, and interim and final reports. In the development part, we start to bring life to our project. This part includes things like data collection, data processing, model preparation, and model training. Lastly, we have the testing and publishing part. In this part, we have the necessary tests for our project and the publication of the website.

Throughout the project, our final goal is to implement a whole system. This system, in short, will be a web browser that uses a machine learning platform in sync with itself to display the results given by the platform.

DETECTING SOCIAL BOTS USING TWITTER DATA

(ÖZET)

Günümüzde çevrimiçi sosyal ağlar hayatımızda önemli bir yer alıyor. Twitter ise bu çevrimiçi sosyal ağların en önemlilerinden biri. Twitter'da insan kullanıcılar tarafından kontrol edilen hesaplar olduğu gibi bilgisayar algoritmaları tarafından kontrol edilen hesaplar da var. Bilgisayar algoritmaları tarafından kontrol edilen hesaplara bot hesap deniyor. Bot hesaplar faydalı olabileceği gibi zararlı da olabiliyor ve bu, botu oluşturan kişinin botu oluşturma amacına göre değişiyor. Zararlı bot hesaplar sahte haberler paylaşmak, bot ağları kurmak, e-dolandırıcılık, politika kampanyaları, bir hesabın takipçi sayısını arttırmak gibi işlemler için kullanılıyor. Bizim amacımız ise, insanların bir hesabın bot olup olmadığını öğrenmek istediklerinde kullanabilecekleri bir sistem oluşturmak.

Bizim sistemimiz, sosyal botları tespit edebilmek için makine öğrenmesi algoritmaları kullanacak. Birbirinden farklı algoritmaları test edeceğiz ve en iyi sonuçları veren algoritmayı, bot tespit etme sistemimizde kullanacağız. Algoritmalarımızı eğitmekte, geçmiş dönemde yapılan çalışmalarda kullanılan veri kümelerini de kullanarak geçmiş dönemdeki çalışmalardan daha iyi sonuçlar elde etme amacında olacağız. Kullanılacak veri kümelerinde, Twitter kullanıcısının hesabının verileri ile kullanıcının attığı tweetlerin verileri bulunmakta olacak. Halihazırda veri kümelerinde bulunan özellikler ile birlikte bu özelliklerden yeni özellikler de elde edip algoritmalarımızı eğitmekte kullanacağız.

Twitterdaki sosyal botları tespit etme sistemimiz 3 kısımdan oluşuyor. Sistemimizin ilk kısmı, ulaşabileceğimiz en iyi sonuçları veren bir makine öğrenmesi algoritmasının tasarlanması. Bu makine öğrenmesi algoritması, Twitterdaki sosyal botları tespit etmede kilit nokta olacak. Bundan sonraki adımda, bir internet sitesi kuracağız. Bu internet sitesi, bot olup olmadığı bilinmek istenen kullanıcının hesabının verilerini ve yine aynı hesabın tweetlerinin verilerini toplayacak. Son kısımda ise internet sitesi tarafından toplanan bilgiler, bir makine öğrenmesi platformuna gönderilecek. Bu makine öğrenmesi platformundan gelen sonuçlar ise bilgilerin toplandığı internet sitesinde kullanıcıya gösterilecek.

Geçmiş çalışmalardan daha farklı özellikler oluşturup modellerimizde kullanarak, sosyal botları tespit etmede daha yüksek isabetlilik sonuçlarına ulaşacağız. Modellerimizin hassaslık ve özgüllük sonuçlarına yer vereceğiz. Sistemimizi daha genelleştirilebilir yapmak için birbirlerinden farklı veri kümeleri kullanacağız. Aynı zamanda kullanıcılar için kolay kullanılabilen ve özgün bir arayüz oluşturarak kullanıcılarımıza iyi bir kullanıcı deneyimi sağlayacağız.

Sistemimizi kurarken işlevsel olan ve olmayan gereksinimleri karşılayacağız. İşlevsel olmayan gereksinimlerden güvenilirlik, kullanılabilirlik, performans, sürdürülebilirlik, ölçeklenebilirlik, taşınabilirlik, desteklenebilirlik, güvenlik gibilerini karşılarken aynı zamanda kullanıcıların iyi bir deneyim yaşayabilmesi için sistemin işlevsel gereksinimlerini de karşılayacağız. Bu gereksinimler kullanıcının botları tespit edebilmesi, sistemin basit bir arayüzü olması, sistemin uygun zamanda cevap verebilmesi gibi gereksinimler olacak.

Sistemimiz; yazılım geliştirmek için laptoplar, Python, HTML, kodlama dillerini geliştirmek için geliştirme ortamları ve algoritmamızı eğitmek için veri kümelerine ihtiyaç duyuyor. Projemizi 4 ana kısma ayırdık. Bu ana kısımlar başlama, dizayn, tasarım ve test & yayınlama aşamalarından oluşuyor. Projemizin başlama kısmında, sistem dizaynları gibi projenin en başında yapılması gereken işler yer alıyor. Dizayn kısmında ise sistemimiz için gerekli algoritmalar, internet sitesi, raporlarımız gibi öğrenmemiz gereken konular var. Tasarım kısmında ise projemizi hayata geçirmeye başlıyoruz. Bu kısımda veri toplama, veri işleme, model hazırlama, model eğitme gibi bölümler var. Son kısım ise test ve yayınlama kısmı. Bu kısımda ise projemiz için gerekli olan testler ve internet sitesinin yayınlanması var.

Projedeki amacımız tüm bir sistemi kendimiz sıfırdan oluşturmak. Bu sistem, kısaca, makine öğrenmesi platformu tarafından verilen sonuçları gösterebilmek için bu platformla senkronize çalışan bir internet sitesi olacak.

Contents

1 Introduction and Project Summary……1
1.1 Engineering Standards and Multiple Constraints……3

2 Comparative Literature Survey……..4

3 Developed Approach and System Model……...8
3.1 Data Model……..8
3.2 Structural Model ……...11
3.3 Dynamic Model ……27

4 Experimentation Environment and Experiment Design……...30
4.1 Evaluation Metrics and Selected Classifiers……...……..….30
4.2 Choosing the Model and Data Collection.……31

5 Comparative Evaluation and Discussion……..32
5.1 Language-Agnostic Approach……32
5.2 One-Classification Approach…….33
5.3 Novel Hybrid Approach…….34

6 Conclusion and Future Work……43

7 References ……44

1 Introduction and Project Summary

With the rise of the internet, social media has become the main source used by people for communication, entertainment, collaboration, content creation, and general socializing. Some of the latest statistics on social media show that in 2021, out of approximately 7.9 billion people in the world, there were 3.78 billion worldwide social media users. As it is in these statistics, the total social media users are equal to a total of approximately 48 percent of the human population. This number is expected to only grow further in the upcoming years [1]. As social media grows even further, its influence on life and general human behavior also increases. An experiment that was done in [2] to get an idea of how often people fact-check under different circumstances, used incentivized tasks of real effort to conclude that when people feel that they are in the presence of other people, they fact-check less likely compared to when they are alone. They also found out that giving a warning to people before evaluating their decision, increased the rate of fact-checking in social settings. This poses a bit of a problem as since social media is such a substantial aspect of our lives, the amount of uncontrolled data that is posted is very large. For example on Twitter, it is stated in [3] that the data gathered in May 2020 shows that around 6000 tweets are tweeted on average; every second. This equates to around 360000 tweets per minute, 518.4 million tweets per day, or 190 billion tweets per year.

Besides regular social media users' activity, the amount of big data generated on social media on a daily basis inevitably leads to fabricated and malicious content. This type of content is generated by social media bot accounts. These bot accounts are accounts that are fully or partially controlled by a computer algorithm and their purpose can range from tweeting simple misinformation to fabricating propaganda for terrorist attacks. Even though it should be noted that not all bot accounts are malicious, we will mostly be touching on the malicious ones and revolve our problems around them. In [4], Paw Research Center conducted research involving the 2315 most favored websites and looked into around 1.2 million tweets that were sent by English language users. These tweets all contained links to one of the 2315 websites and were gathered in a six-week span in the summer of 2017. Their results showed that autonomous accounts played a key role in spreading links to the popular websites chosen. Out of all the links to the popular websites, 66 percent of links were shared by autonomous bot accounts, rather than real users. It is clear that a lot of the content generated on social media is created by social media bots. This introduces us to the problem that since most of the data we perceive in social media is generated by bot accounts, it is not quite clear which information is trustworthy or not. In [5], it is stated that this massive digital misinformation spread is recognized as a crucial threat to democracy. Cognitive, social, computer and communication scientists are researching the complex reasons for the spread of misinformation. On the other hand, social platforms are starting to establish countermeasures. Detection of such social bots is the first step and the key to dealing with the spread of misinformation and malicious content. Facebook stated the removal of 1,5 billion fake accounts in Q2 of 2020, less than they removed in Q1 of 2020, 1.7 billion accounts. Guy Rosen, who is a VP of Integrity on Facebook, said ”We estimate that fake accounts represented approximately 5 percent of our worldwide monthly active users (MAU) on Facebook during Q2,” [6].

Algorithms created to detect social bots are the most prominent way of dealing with misinformation and malicious content that is spread on social media. Our project will focus on the problem that is social bots that are used to spread such content and will devise a machine learning algorithm that will be used in the detection of social bots. For the social media platform of our choosing, we will use Twitter. With its active user count increasing since 2017 and having reached a total of 330 million active users by 2021, Twitter is one of the most popular online social networks [7]. As it usually comes with having such a large user base, Twitter also has many bot accounts. Even an algorithm that has a high detection rate accuracy-wise, can oversee millions of undetected bot accounts. Considering this information, it isn't surprising that [8] estimated the total active bot account percentage on Twitter to be 9–15 percent in 2017. We will develop a system that detects these accounts and warrant information about them to the users.

The general system we will be implementing for bot detection is going to be supervised machine learning algorithms that detect bot accounts. The algorithms we are going to use will do binary classification on the data and depending on if the account is founded to be a bot or not, it will provide a binary response as in 1 (if the account is deemed to be a bot) and 0 (if the account is deemed to be a regular user). We will be using different machine learning algorithms such as random forest, support vector machines, etc., and use cross-validation to get more information on how the algorithms would generalize on independent sets of data. We will also investigate the models' performance by different metrics like accuracy, precision, sensitivity, specificity, and f1 score.

To train the machine learning algorithms, we will use different types of bots to create a more generalizable model that is suitable for different kinds of situations that might be required in bot detection. The example data set we will use is the Cresci-2017 data set that is used in previous works of supervised machine learning algorithms in social bot detection. This data set contains information about 14.368 Twitter accounts; 3.474 human accounts, and 10.894 bot accounts with their tweet data [9]. Finally, we will create a website that will gather Twitter data, which will then send this data to a machine learning platform. There, the prediction results of the algorithm will be calculated and then the results will be shown to the user on the same website.

For the problem that is social bots; we try to create a novel approach that detects social bots using supervised machine learning algorithms, combined with an easy-to-understand presentation of the results for a better user experience. Particularly, the contributions of our project can be listed as follows:

1. Feature extraction on a new set of features.
2. Higher results of accuracy, precision, and recall compared to past studies.
3. Unique user interface that is easy to understand and use.
4. Explanations of our detection method and its decisions for product integrity.

The report from here on out is constructed as follows: Section 2 discusses the comparative literature survey we have done throughout our research. Section 3 presents our developed approach and system model, specifically our data, structural and dynamic model, and interface in 3.1, 3.2, and 3.3. This is followed by the details of our experimentation environment and experimental design in 4 while our comparative evaluation and discussion are given in 5. Finally, we discuss our conclusions and future work in 6.

1.1 Engineering Standards and Multiple Constraints

In the process of our project, we wanted to build a standard that generalized the research done on the topic of social bot detection. To do this, the literature review was the most important part as we needed to learn to be able to understand what kind of processes different researchers went through to accomplish the task at hand. With this, we were able to devise a standard revolving around social bot detection that required specific steps to get to a final result. Throughout the report, these steps will be clear as they are explained thoroughly but to give the idea in short; the general standardized steps we used in our project were:

1. Acquisition of dataset
2. Pre-processing
3. Feature Extraction
4. Model selection and evaluation
5. Conclusion

As it'll be seen in 3.2, we applied this standardized approach to different methods and tried to find the best possible social bot detection model we could achieve. Through these different approaches, there were also constraints that we had to keep in mind. One of the important constraints came from the codes we used and that the Cresci-17 database we used in our project was huge in comparison to what some of the python libraries we used could handle in memory. Since each tweet and their attributes were a row in a CSV file, the Cresci-17 dataset will be discussed more in detail in 3.1 so for now just knowing this information is fine, because of this, the libraries couldn't handle the huge memory requirements when there were just too many tweets, rows, to process. Because of this, we reverted to using only some parts of the dataset by filtering while also trying to create an adequate balance between bots and humans. Another came in the form of policy, regulations and legal considerations for social bot detection as the data we'll be using, when the model is up and running on our website and we're detecting bots from live data, is actual data from the real world, Twitter, which could be sensitive. Twitter already has policies regarding the shared data and these policies were shared with us as we were given access to use the Twitter API. To get access to the Twitter API, We had to explain what we would be doing with the fetched data, if we would be altering the data into any forms, if the data would be shared, and if there would be any commercial use, etc. The constraints and policies that Twitter uses indicate that we're only able to use the data and no other operations such as storing, commercial use, altering, etc. Twitter API also had constraints in itself where the newest version of it, v2, was very much lacking in data fetching as most of the features that were in the Cresci-17 dataset weren't accessible with the newest version. While building our project, we also had to keep this in mind. The last constraint we had was in the form of functionality as we chose to design our machine learning model as a binary classifier that classifies users as a bot or a human. Even though this gave us the ability to give more determinant answers when detecting bots, it was also a constraint we had since we decided to not use things like confidence intervals for our results.

2 Comparative Literature Survey

Social bots come in many different forms and there have been many machine learning techniques to try and detect such types of bots. In this section, we briefly review some of these bots, their types, and recent studies that have been devised in their detection problem of them.

As previously said, not all bots are used for malicious activity. There are many bots on Twitter that are designed to do a specific thing, and in their context, might even be very helpful. The DownloaderBot Twitter account, for example, is a bot designed so Twitter users can download videos and gifs shared on the platform. There are many other social bots that don't try to cause any harm. In [10], the existence of social bots is discussed by viewing them by metrics concerning social media (altmetrics). They believe that mentioning a scientific article on Twitter creates a bigger and broader impact than it does when citations are used. It is concluded that social bot accounts on Twitter create a sizable number of tweets to or in response to scientific articles. Some definitions of Twitter bots are discussed and a distinction is proposed on the different amounts of interaction these bots have with tweets. While some only tweet bibliographic information, some of them go to the lengths of discussing scientific articles with normal users. While bibliographic information might be useful, a discussion with a Twitter bot about a scientific article would make a real user have their judgment affected. It is clear bots can be used with different intentions depending on how they are created, but bots that create malicious and harmful content are the bots we will be discussing the most.

One of the most common bots that we come across today is the fake follower. These types of accounts are accounts that are explicitly created to increase the total follower count of an account. Their danger comes from the fact that since they can inflate follower count, they create falsely perceived influence on Twitter. This type of influence can affect many things such as politics, economy, society, etc. [11]. It is also stated in [11] that there have been media reports that state high inflation of account followers of celebrities, politicians, and very popular brands that created suspicion. In [12], it is stated that misinformation spread is a major danger during the times of COVID-19 so we don't face an infodemic. Thus, stating the identification of influential content that needs to be exposed is also very important. Another type of bot is the spam bot. Spammers are accounts that disturb regular users with high numbers of tweets. These tweets are named spam and they have many different purposes such as product advertisements, fake news, unwanted tweets, etc., [13]. The research that was done in [5] analyzed 400 thousand articles that contained a total of 14 million messages during 2016 and 2017. It was found that social bots had a major part in spreading low-credibility sourced articles. Bots amplified the spread of the article before it went viral. Bots also targeted high follower count users via mentions and replies. Also stated in [5], a study analyzing 14 million tweets from the time of the 2016 U.S. presidential election suggested that social bots had a big role in spreading low-credibility content. Results of these studies suggest that limiting social bots would be an effective way of handling misinformation. There are also bots that can form a network. These types of bots are called Botnets. These are a remotely controlled network of computers that behave in a way that resembles the normal traffic activity of a social circle. It is because of this that their behavior is hard to detect by current Intrusion Detection Systems (IDS). Computer networks of today are faced with increasing threats from botnets' malicious activity [14]. Currently, there are also many innovative and sophisticated implementations

of social bots. These accounts are created with the idea to resemble regular user (human) accounts and mimic their behavior on a meta-data level. This behavior mimicking can go as far as to communicate with other users. Even humans may be misled by these sophisticated techniques [15].

With bot accounts like the ones that can even trick humans, it becomes very important to devise algorithms that can detect these social bots. Machine learning has been an extensively used field in social bot detection. Here, we will discuss some of the approaches that have been devised in the problem of social bot detection.

Even before discussing the usage of machine learning algorithms in social bot detection, there were early ideas on how to detect social bots. In, [16] a social bot detection method using crowdsourcing was proposed. This method was one of the very early methods of bot detection. The method included creating a website for test subjects to see if they can discriminate the bots from the data sets they use. Test subjects were divided into three groups named experts, turkers, and sociology undergraduates. Each tester evaluated more than 10 different profiles and all profiles were evaluated by different test subjects from every group. This method was quite effective; although it had problems in scalability, cost, and privacy. Because of this, different machine learning methods are more commonly used in the detection of social bots.

Machine learning algorithms used in social bot detection have 3 different types of techniques implemented: Supervised, semi-supervised and unsupervised. Supervised machine learning techniques are techniques where the data is labeled, semi-supervised machine learning techniques are techniques where a small amount of data is labeled while a larger amount of data is unlabeled, and unsupervised machine learning techniques are techniques where no data is labeled and ground truth data is used for evaluation [17].

The most commonly used machine learning technique in social bot detection is the supervised machine learning technique [17]. In [18], a supervised technique was used. It was assumed that bots behave differently in fundamentals when compared to humans. These differences were categorized under technical, purpose-related differences and the approach to bot detection was based on this assumption. The approach combined features that are account-based and tweet-based. Account-based features included behavioral and emotional features while tweet-based features included user profile and user profile name features. Different classifier algorithms such as random forests, multilayer perceptrons, logistic regression, support vector machines, and AdaBoost were used. The best performance was achieved by the classifier AdaBoost with precision going to 0.9969 with an accuracy of 0.9881 on the dataset Cresci-2017. In the efforts to create a bot detection method that is easier to be understood by a human and to achieve higher accuracy compared to different classification methods, [19] proposed a contrast pattern-based approach to detect bots on Twitter. To achieve this, they suggested a different feature model. Their proposed model combined sentiment analysis that they extracted from the Twitter accounts' text and the frequency data in combination with additional features related to the time, and date features of the account and its tweets. It was shown that sentimental analysis is suitable as a bot detection method and is usable for detecting Twitter bots in different languages through their correlation analysis. Also by their classification results, it was shown that their newly suggested model proved to be better than other contrast pattern-based approaches. In [20], a system called User Behavior Analytics-based Compromised Account Detection (UbCadet) was proposed. This system

created a behavioral profile for a Twitter account and according to this profile, the tweets of these users were converted to their tweet patterns. The K-Nearest Neighbour algorithm was then used to detect if these tweets were malicious or not. In addition to these, random forest and ensemble machine learning methods were also experimented with. An assessment of the account was done based on the frequency of the malicious tweets. Results showed that UbCadet's approach to bot detection was better than earlier works of behavioral profiling as it could detect more than %90 malicious tweets. A new set of features, namely graph-based and content-based features, were proposed by [21]. They derived the graph-based features as social graphs that were built by users on Twitter while content-based features were driven from the accounts' tweet texts. These features were then applied to seven different machine learning algorithms; namely decision tree, support vector machine, random forest, k-Nearest neighbor, naive bayes, logistic regression, and XGBoost, and got the best results in the random forest algorithm. Their results were compared with other state-of-the-art approaches and they found out that this set of features is suitable for giving better results compared to those approaches in bot detection on Twitter. In [22], it was proposed that the difference between real users' and bots' posts are distinguishable. Because of this, a classification model to detect bot accounts only using the accounts' tweets was proposed. They did this by combining all the tweets of a user into one single document and used NLP feature extraction methods tf-idf, bigram, and Word2Vec. They used the model's logistic regression, ADA Boost, XGBoost, and Random Forest with additional classification to distinguish bot accounts from real users. XGBoost was the best performer, detecting bots with an accuracy of 95.55. An approach that affiliated the tweets about natural disasters with a credibility system was proposed in [23]. Credibility-Analysis-System was designed to rank the tweets related to their credibility. Different state-of-the-art machine learning classifiers such as naive bayes, support vector machine, and random forest were used. Also, the authors used a real-time data set that was gathered during the Gaja cyclone storm. The random forest algorithm proved to be the best classifier with an accuracy of %87.

Semi-supervised machine learning techniques are also useful in the detection of social bots. In platforms like social networks where labeling is time-consuming and costly, semi-supervised machine learning techniques are a very good alternative [17]. In [24], it was suggested that even though there has been an increasing interest to design botnet detection methods, these techniques generally relied either on the use of social behavior or social graph to detect bots from actual users. It was also pointed out that even though these methods were promising, they could not converge fast and detect bots that are designed to mimic the worldly behavior of users. To tackle this issue, they proposed SocialBotHunter, which is a semi-supervised technique that uses both social behavior and social graph information equally to detect botnets in Twitter-like social networking services. Another idea was stated in [25] saying that both legitimate users and spam users knew the state-of-the-art APIs used by Twitter in the context of written and displayed information. Since Twitter has its unique features, they argued that it's not possible to detect spam accounts on Twitter using the general tools for detecting spam accounts. Therefore, they suggested a spam detection mechanism that was designed uniquely for Twitter called TwitterSpamDetector. TwitterSpamDetector identifies spam using characteristics specific to Twitter. They used the Naive Bayes method to train on these special features and reach precision, and sensitivity of 0.943 and 0.913. In [26], it was suggested that even though many bot detection methods are used with the help of artificial intelligence, the reasoning behind the findings and their characterization is not always transparent, which lacks ethical duty. These algorithms are generally not publicly shared and because of this, their real

efficiency is in question. To tackle this issue, they proposed a bot detection approach that is interpretable, transparent, and responsible. They combined an illustratable machine learning framework with a strong crowdsourcing mechanism for user feedback to create a public web service. They also created a new annotated data set to use in their service and share publicly for public use. It was stated in [27] that many approaches to bot detection were based on the characterization of that specific user and that these approaches didn't take into account the interactions this user would have with other users, so they present a hybrid approach. Assuming the user can avoid detection with features related to their profile data, they proposed an approach including both community-based features and general user account features. They proposed 6 new features, and 2 redefined features and used three classifiers; namely decision trees, random forests, and bayesian networks on a real data set. They also tested the effectiveness of different features in the detection of bots and found that community-based features are more effective compared to metadata-based features.

Even though most of the bot detection techniques rely on supervised and semi-supervised approaches, there are also unsupervised approaches. In [28], it was suggested that the recent increase in social interactions had attracted cybercriminals who use spam as a way of engaging in malicious activities. It was also suggested that detecting these spammer networks is crucial on the road to identifying spam accounts on their own. They tackled this issue by proposing an approach that detects Twitter spammers according to their resemblance to already existing spam accounts. With this, they tried to improve on other research by not specifically targeting a category of spammers. They used Principal Component Analysis (PCA) and K-means on 200.000 accounts selected randomly from 2 million tweets for the detection of spammer clusters. These clusters were then used as ground-truth to train 3 classifiers; namely fandom forest, multilayer perceptron, and support vector machine with random forest having the best results. A study in [29], used Concurrent Content Injection Detection (CCID) in identifying botnets. Their method, CCID, detected bot accounts assuming that more than one account had activities alike synchronously. They used thresholds to make sure that the assumptions didn't merely just catch coincidences and that the correctness of the assumptions would be a powerful indicator of detecting if bot accounts were managed by the same software or operated by the same person that manages botnets with software. They analyzed 720 million tweets to discover 73 thousand Twitter accounts that behaved synchronously. They also used crowdsourcing information and correlated it with their findings.

There have also been studies that challenged the idea of bot detection in the sense that the importance of a social account being a bot is questioned. In, [15] a difference in perception in social bot detection is proposed. Based on two case studies involving the Botometer (formerly BotOrNot) [30] and Germany's September 2017 general election, a change of perspective is achieved. They proposed that campaign detection and automated strategy are a lot more important in security against malicious attacks than actually detecting if that malicious attack is done by a bot or not.

3 Developed Approach and System Model

In the section, we discuss our developed approach for the problem of social bot detection and discuss our system model using different modeling approaches. To do these, we explain our data model in 3.1, our structural model in 3.2, and our dynamic model in 3.3. In 3.1, we discussed our data model which includes extensively explaining the Cresci-17 dataset we've used in our project and our overall data model of the system we've created. In 3.2, we discussed our structural model which includes two different feature extraction methods we tried and a final method that creates a hybrid approach with the first two approaches in combination with other features. The final method is also the one we use for bot detection. In 3.3, we discuss the overall dynamic model of the system in addition to our interface discussions.

3.1 Data Model

For our data model, we'll be discussing two things. Firstly, we'll discuss the Cresci-17 dataset we'll be using to create our model and secondly, we'll discuss the general data model of the final system we've created.

3.1.1. Cresci-17 Dataset

The Cresci dataset is a publicly available example dataset used in previous works of supervised machine learning algorithms in social bot detection. The dataset contains information about a total of 14.468 accounts; 3474 human accounts, and 10,894 bot accounts. You can see a picture of what the dataset looks like in Figure 3.1.

Figure 3.1: Format of the Cresci-2017 Dataset

As can be seen from the picture, human accounts are named as genuine users and the bot accounts are split into three categories which are traditional spambots, social bots, and fake followers. This is an important concept in this dataset as since there are different types of bots, it gives us the ability to generalize our model for different scenarios that might happen in live bot detection. For each type of user, bot, and human, the dataset contains account information. It also contains tweet data for most of the accounts. For our models, we'll be doing feature extraction on both account and tweet features. Because of this, in feature extraction parts, we'll be extracting features from users that contain tweet data.

Both account and tweet information in the dataset is given in the form of CSV files and the contents for these files change depending on if they are the "users" or "tweets" files. Every user and tweet was designed to be a row in the CSV file while the attributes of the users and the tweets were given in the columns. You can see the ER diagram in Figure 3.2 which shows the attributes a user and their tweet have while also showing the connection between them.

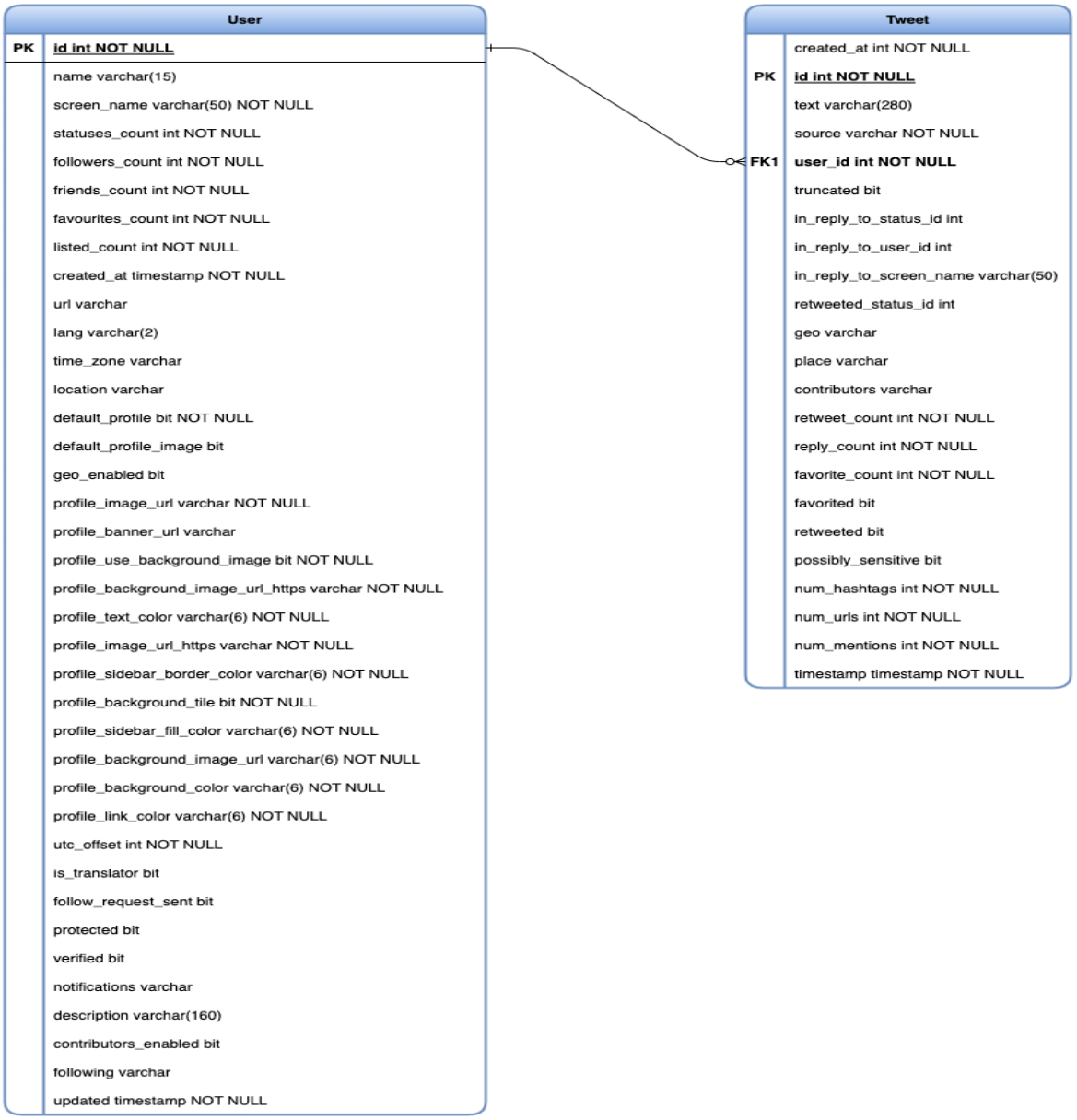


Figure 3.2: E-R Diagram of the Cresci-2017 Dataset

The “users” files contain different Twitter users with their id, which is being shown as the primary key in the diagram, being the main discriminator between them. Each user has many different attributes and these attributes can be used to extract features directly by taking the attribute or indirectly by doing some operations on the attribute(s).

The “tweets” files contain all the tweets of all the users given in the “users” files. All of the tweets differentiate from each other using their id, which is again shown as the primary key in the diagram. In addition to this, the tweets also have an attribute called user_id, which is shown as the foreign key in the diagram, which relates each tweet to the user. This’ll allow us to match each user with their tweets and extract the features from there. Also, just like the users, their tweets have many attributes that can be used to extract features directly or indirectly.

Going through the dataset and creating an ER diagram was very helpful as we were able to determine how the dataset attributes were constructed and what to do when preprocessing comes. Determining the types of the attributes, which of them can be null etc. made it convenient for us to do preprocessing.

3.1.2. Overall Data Model

In Figure 3.3, we represent our overall data model as a component diagram.

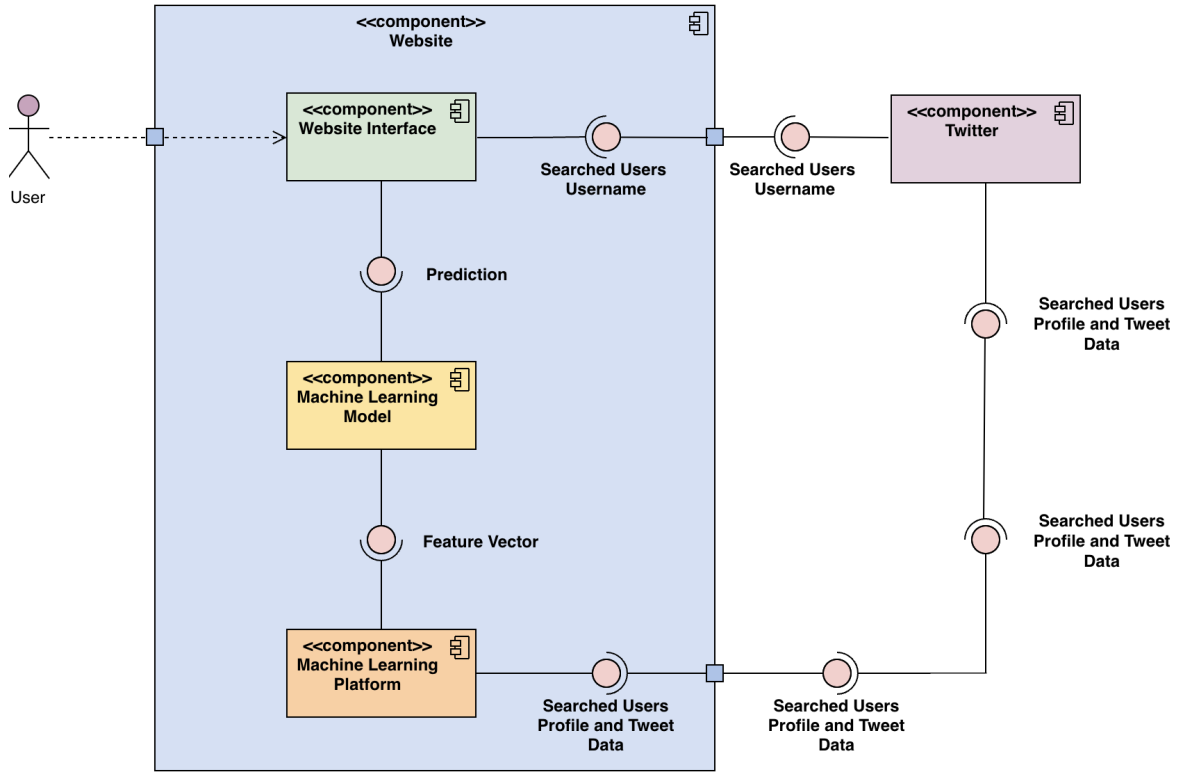


Figure 3.3: Component Diagram of the Implementation

In the diagram, our system as a whole can be seen. We built a website using Flask, which is shown as the component “Website”, to be able to embed the machine learning model we have and use it. The website contains an interface, the component “Website Interface”, where the users can enter the Twitter account they want to do predictions for by entering the account's username. The account’s username is then sent to Twitter, the component

“Twitter”, in the form of queries to fetch the data of the account. Twitter then returns the profile and tweet data of the account to the machine learning platform, the component “Machine Learning Platform”. Since we’re using Flask, we’re able to create a machine learning platform, which is python code to do certain operations, which creates the feature vector of the account entered by the user. This feature vector is then transferred to the machine learning model, the component “Machine Learning Model”, for the prediction to be done. Lastly, the machine learning model sends the prediction back to the website interface where it’s displayed to the user.

3.2 Structural Model

To form our model's basis, we sought past research and found two papers that would be used for developing our approach to distinguish between human and bot accounts. These two papers applied Language-Agnostic and One-Classification methods to the Cresci-17 dataset and found social bots this way. By going through these papers, we were able to create the re-do versions of these approaches, while also adding a bit of our intuition. By taking inspiration from these papers, and some others which are mentioned further in the report, we were then able to create our own bot detection model, which is a Novel Hybrid approach that detects social bots using many different feature categories. For the two approaches we sought out, we’ll briefly explain the idea behind both of them while also giving the pseudocode for the feature extraction we used. We’re only giving a brief explanation and focusing more on the pseudocode part as these approaches are thoroughly discussed in their own papers. For our Novel Hybrid Approach though, we’ll be discussing it very exhaustively, trying to give as much detail as possible. This is obviously our newly developed, novel, approach so it’s also the approach that we’ll be using to select the model to use in our system.

3.2.1. A Language-Agnostic Approach

In [18], they assume that bots are fundamentally different from humans and state two categories of distinguishment: Technical and Purpose-related differences. For the technical differences, they state that since bots are computer programs, they don’t have the limitations humans do, therefore having different behaviors regarding the published contents orientations and timing. They also state that as another technical difference, it’s difficult for bots to imitate the behavior of humans. They say this by stating that simulating human inadequacies might be possible but it’s difficult to accurately imitate human behavior and that a large-scale statistical analysis would be needed. Because of this, they assume that simpler methods are used when creating bots. For the purpose-related differences, they assume that each bot has a clear objective, spreading misinformation, spam, etc. They state that bots, for their objective, “agenda”, use methods like URLs and hashtags which can be exploited.

After these assumptions, they extract their features accordingly to prove them. They mainly group their features into 2 categories, account-based, and tweet-based features. In the account-based features, they use the features that are directly provided by Twitter and also extract some other features that can be derived from them.

In our implementation of the approach, we used the below account-based features.

- Default Profile: Indicates whether the user has altered their profile.
- Geo Enabled: Indicates whether the user enabled adding geographic information to their tweets.
- Protected: Indicates whether the account is in protected mode.
- Is Verified: Indicates whether the account is verified.
- Friends Counts: Number of accounts that follow the user.
- Followers Counts: Follower count of the account.
- Listed Count: Number of public lists the account is an associate of.
- Statuses Count: Tweet count of the account.

The above accounts-based features are the features that are taken directly from the Twitter API. In our implementation of this model, we removed the “Profile Use Background Image” feature since Twitter API doesn’t support this feature anymore. In addition to these features, there were also ones derived from them. The below list shows these features.

- Screen Name Length: Length of the users' screen name.
- Username Length: Length of the users' username.
- Screen Name Digits: Number of digits in the users' screen name.
- Levenshtein Distance: Levenshtein distance value between the users' username and screen name.

The above accounts-based features were derived from the features that can be taken directly from the Twitter API. In our implementation, we removed two derivable features from the original model. These features were user “Username Unicode Group” and “Screen Name Unicode Group”. We removed these features because they added too many features to the model. In the original approach, they added a column for every Unicode group, 105 Unicode groups for each feature, for a total of 210 Unicode groups.

For the content-based features, they derived features from the content of a user. For this purpose, they used tokenization of the users' tweets and tweet attributes that the Twitter API provides. Under the content-based features, they created two categories called behavioral features and core-content features. For the behavioral features, they assume that bots and humans behave differently. To model the difference between them, they calculate properties on the tweet data using statistical analysis with functions such as standard deviation, mean, median, average, etc.

In our implementation of the approach, we used the below behavioral features.

- Time Between Tweets: Average time between the publication of two tweets.
- Time Between Retweets: Average time between the retweets of the account.

In our implementation, we removed the tweet rate feature, which was a feature that showed the average tweets a user posted per day, because we felt like the number of tweets a user posts a day, didn’t help in the way of detecting whether or not they were a bot.

For the core-content features, they assume that emotions and intentions can be derived from the content of a tweet. They also assume that sentiment analysis can't be precise for all the languages that can be used to write a tweet. To derive emotional aspects from the tweets, they use some basic emoticon aspects; deriving four emoji features. These features aim to find the number of occurrences of classic, kaomi face, line art, and other emojis. In our implementation, we take all of these emojis as a single group and find the total number of emojis in a tweet. In addition to these, they also use a “Number of Tokens”, “Number of Hashtags”, “Number of Tokens Without Hashtags and URL Symbols”, “Number of URLs” and “Special Character Repeat Rate” features. They say that to some extent, they’re able to model the tweet size with the “Number of Tokens” feature. So, we just directly use “Tweet Size” as a feature instead of “Number of Tokens”. We also add the “Number of Hashtags” and “Number of URLs” features to our implementation. In our implementation, we didn’t tokenize the tweets. Because of this, we didn’t use the “Number of Tokens Without Hashtags and URL Symbols” and “Special Character Repeat Rate” features. Instead of these, we use different results derived using the statistical analysis functions on the mentioned features. To give an example, by taking the total number of emojis in each tweet, we can then use statistical analysis functions like average, median, etc. to get new features.The complete list of core-content features used in our implementation can be seen below.

- Emoji Count Average: Average emoji count per tweet.
- Emoji Count Median: Median of the emoji count of each tweet.
- Emoji Count Standard Deviation: Standard deviation of the emoji count of each tweet.
- Emoji Count Skewness: Skewness of the emoji count of each tweet.
- Emoji Count Kurtosis: Kurtosis of the emoji count of each tweet.
- Tweet Size Average: Average tweet size per tweet.
- Tweet Size Median: Median of the tweet size of each tweet.
- Tweet Size Standard Deviation: Standard deviation of the tweet size of each tweet.
- Tweet Size Skewness: Skewness of the tweet size of each tweet.
- Tweet Size Kurtosis: Kurtosis of the tweet size of each tweet.
- Number of Hashtags Average: Average number of hashtags per tweet.
- Number of Hashtags Median: Median of the number of hashtags of each tweet.
- Number of Hashtags Standard Deviation: Standard deviation of the number of hashtags of each tweet.
- Number of Hashtags Skewness: Skewness of the number of hashtags of each tweet.
- Number of Hashtags Kurtosis: Kurtosis of the number of hashtags of each tweet.
- Number of URLs Average: Average number of URLs per tweet.
- Number of URLs Median: Median of the number of URLs of each tweet.
- Number of URLs Standard Deviation: Standard deviation of the number of URLs of each tweet.
- Number of URLs Skewness: Skewness of the number of URLs of each tweet.
- Number of URLs Kurtosis: Kurtosis of the number of URLs of each tweet.

Pseudocode for the Language-Agnostic Approach Feature Extraction for User Features:

```
def string_length(string):
    return len(string)
def string_digit_count(string):
    count ← 0
    for character in string:
        if isdigit(character):
            count ← count + 1
    return count
def lev_distance(username, screenName):
    return lev(username, screenName)
// accounts is read from csv file
df["default_profile"] ← account["default_profile"]
df["geo_enabled"] ← account["geo_enabled"]
df["protected"] ← account["protected"]
df["verified"] ← account["verified"]
df["statuses_count"] ← account["statuses_count"]
df["followers_count"] ← account["followers_count"]
df["friends_count"] ← account["friends_count"]
df["favorites_count"] ← account["favorites_count"]
df["listed_count"] ← account["listed_count"]
df["label"] ← account["label"]
df["name"] ← account["name"]
df["screen_name"] ← account["screen_name"]

df["user_name_length"] = apply string_length to df["name"]
df["screen_name_length"] = apply string_length to df["screen_name"]
df["screen_name_digits"] = apply string_length to df["screeen_name"]
df["lev_distance"] = apply lev_distance to df["name", "screen_name"]
return df
```

Pseudocode for the Language-Agnostic Approach Feature Extraction for Tweet Features:

```
tweets = get_user_tweets(user_id)
tweet_time_array, retweet_time_array, emoji_count_array, tweet_size_array ← []
hashtags_count_array, urls_count_array ← []
tweet_time_difference, retweet_count, retweet_time_difference, tweet_count ← 0
emoji_count, tweet_size_count, number_of_hashtags, number_of_urls ← 0
for tweet in tweets:
    append tweet.time to retweet_time_array
    if isRetweet(tweet):
        retweet_count ←retweet_count +1
    else:
        tweet_count ← tweet_count + 1,
        tweet_size_count ← tweet_size_count + length of tweet.text
        append length of tweet.text to tweet_size_array
        number_of_hashtags ← number_of_hashtags + tweet.num_hashtags
        append tweet.num_hashtags to hashtags_count_array
```

```
16:             emoji_count ← emoji_count emoji count of tweet.text
17:             append emoji count of tweet.text to emoji_count_array
18:             number_of_urls ← number_of_urls + tweet.urls_count
19:             append tweet.urls_count to urls_cout_array
20:
21:     total_tweet_time_difference_in_second ← total difference of tweet_time_array
22:     total_rt_time_difference_in_second ← total difference of retweet_time_array
23:
24:     // df is dataframe
25:     df["tweet_rate"] ← total_tweet_time_difference_in_second / tweet_count
26:     df["retweet_rate"] ← total_rt_time_difference_in_second / retweet_count
27:
28:     df["emoji_count_average"] ← emoji_count / tweet_count
29:     df["emoji_count_median"] ← median of emoji_count_array
30:     df["emoji_count_stdev"] ← standard deviation of emoji_count_array
31:     df["emoji_count_skewness"] ← skew of emoji_count_array
32:     df["emoji_count_kurtosis"] ←kurtosis of emoji_count_array
33:
34:     df["tweet_size_average"] ← tweet_size / tweet_count
35:     df["tweet_size_median"] ← median of tweet_size_array
36:     df["tweet_size_stdev"] ← standard deviation of tweet_size_array
37:     df["tweet_size_skew"] ← skew of tweet_size_array
38:     df["tweet_size_kurtosis"] ← kurtosis of tweet_size_array
39:
40:     df["number_of_hashtags_average"] ← hashtags_count / tweet_count
41:     df["number_of_hashtags _median"] ← median of hashtags_count_array
42:     df["number_of_hashtags_stdev"] ← standard deviation of hashtags_count_array
43.     df["number_of_hashtags _skew"] ← skew of hashtags_count_array
44:     df["number_of_hashtags _kurtosis"] ← kurtosis of hashtags_count_array
45:
46:     df["number_of_urls_average"] ← number_of_urls / tweet_count
47:     df["number_of_urls _median"] ← median of urls_count_array
48:     df["number_of_urls _stdev"] ← standard deviation of urls_count_array
49:     df["number_of_urls _skew"] ← skew of urls_count_array
50:     df["number_of_urls _kurtosis"] ← kurtosis of urls_count_array
51      return df
```

3.2.2. A One-Classification Approach

In [31], the researchers developed a model by using a one-class classification approach. In one-class classification, the dataset contains only one class. This approach is used when the model tries to find the exceptions that can occur in that specific class, but since there is only one class, the anomalies are rare. Approaching the Twitter bot detection topic, they take into account the fact that the behaviors of bot classes aren't homogenous. They also say that bots behave differently than humans and that is why they use a one-classification approach. By creating a one-class classification approach, they create a model that detects the behaviors of legitimate accounts, which with this, can also detect a deviation from the legitimate user regardless of what type of bot is creating the deviation. For their dataset, they also use Cresci-17, just as in the Language-Agnostic approach. They test their

assumptions with both one-class classifiers and binary classifiers for comparison. For feature extraction, Rodríguez, J. et al. uses the features given in Figure 3.4 [31].

Characteristics	Description	Type
retweets	Ratio between retweet count and tweet count.	Account Usage
replies	Ratio between reply count.	Account Usage
favoriteC	Ratio between favorited tweet and tweet count.	Account Usage
hashtag	Ratio between hashtag count and tweet count.	Account Usage
url	Ratio between url count and tweet count.	Account Usage
mentions	Ratio between mention count and tweet count.	Account Usage
intertime	Average seconds between postings.	Account Usage
ffratio	Friends-to-followers ratio.	Account Information
favorites	Number of tweets favorited in this account.	Account Usage
listed	Number of listed tweets in the account.	Account Information
uniqueHashtags	Ratio between unique hashtag count and tweet count.	Account Usage
uniqueMentions	Ratio between unique mention count and tweet count.	Account Usage
uniqueURL	Ratio between unique urls count and tweet count.	Account Usage

Figure 3.4: Features of the One-Classification Approach

In our implementation of this model, we didn't use unique hashtags, unique mentions, and unique URL features. For their approach, they conducted both one-class classifier and binary classifier experiments while we only conducted binary-classifier experiments since that would be the direction we would be going for our novel approach. We also compared the results of this approach to the Language-Agnostic approach.

Pseudocode for the One-Classification Approach Feature Extraction:

```
tweets ← get_user_tweets(user_id)
posting_time_array ← []
posting_time_difference, retweet_count, tweet_count, reply_count ← 0
favorite_count, friends_count, number_of_hashtags, number_of_urls ← 0
number_of_mentions, followers_count ← 0
for tweet in tweets:
    append tweet.time to posting_time_array
    if isRetweet(tweet):
        retweet_count ← retweet_count + 1
    else:
        tweet_count ← tweet_count + 1
        if isReply(tweet):
            reply_count ← reply_count + 1
        number_of_hashtags ← number_of_hashtags + tweet.num_hashtags
        number_of_urls ← number_of_urls + tweet.urls_count
        number_of_mentions ← number_of_mentions + tweet.mention_count

user_data ← get_user_from_database(user_id)
favorite_count ← user_data["favorites_count"]
friends_count ← user_data["friends_count"]
followers_count ← user_data["followers_count"]

total_posting_time_difference_in_second ← measure_difference(posting_time_array)

// df is dataframe
```

26: df[“retweets”] ← *retweet_count* / *tweet_count*
27: df[“replies”] ← *reply_count* / *tweet_count*
28: df[“favoriteC”] ← *favorite_count* / *tweet_count*
29: df[“hashtag”] ← *number_of_hashtags* / *tweet_count*
30: df[“url”] ← *number_of_urls* / *tweet_count*
31: df[“mentions”] ← *number_of_mentions* / *tweet_count*
32: df[“intertime”] ← *posting_time* / (*tweet_count* + *retweet_count*)
33: df[“ffratio”] ← *friends_count* / *followers_count*
34: df[“favorites”] ← *favorite_count*
35: **return** df

3.2.3. A Novel Hybrid Approach

To create a novel machine learning model that we would be using in our system discussed in 3.1.2, we created a novel hybrid approach to feature extraction that was created with our intuition, in addition to inspiration from [18], the paper discussed in 3.2.1, [27] and [31], the paper discussed in 3.2.2. For this approach, we used features that were derived from both user and tweet attributes, which are called user and tweet features. Our final approach was done in three steps, namely, pre-processing, feature extraction, and post-processing. Each step consists of explanations, pseudocode, and other information regarding the step. When reading, keep in mind that the pseudocodes are continuous and that any variable used in one pseudocode can be used in the latter ones.

1 - Pre-Processing

The pre-processing part of the approach consisted of deciding on which portion of the dataset we would be using to then make sure this part of the dataset would be ready for feature extraction. To create a balance between the bot counts and avoid overfitting, only one portion of each bot was selected. Since genuine accounts, fake follower bots only had one file and traditional spambots only had one file that contained tweets, all of these were selected. This resulted in 3474 genuine accounts, 1000 traditional spambots, and 3351 fake followers. The last thing we had to do was select the social spambots. To create a balance between bots, only one file of the social spambots; social_spambots_2, refer to Figure 3.1, which had 3457 social spambots. In the end, we had a dataset of 3474 human accounts and 7808 bot accounts which added to a total of 11282 accounts. When which portion of the dataset we would use was determined, we now had to make sure it was ready for feature selection. To do this, with the help of our ER diagram in Figure 3.2, we started filling out the null values which would cause trouble in feature extraction. Even though we still haven’t discussed the selected features, which will be discussed just a little later in the report, we already knew what features we would be using as these kinds of decisions run parallel when designing the project in real-time. Because of this, filling the null values was easier as we just needed to fill the attributes that we would be needing in feature extraction. As tweet data of users consisted of bigger CSV files, they were dealt with separately while user CSV files were concatenated before doing the filling. Of the tweet attributes that would be used in feature extraction, only the “text” attribute contained null values which were filled with empty strings. From the user attributes; “geo_enabled”, which was filled with 0’s, “name” and “description”, which were filled with empty strings, were filled. Lastly, all users were labeled with 0 and 1, where 0 was used for humans while 1 was used for bots.

Pseudocode for Pre-Processing:

1: *traditionalBots, traditionalBots_tweets* ← users, tweets files for traditional bots
2: *socialBots, socialBots_tweets* ← users.csv, tweets files for social_bots
3: *fake_followers, fake_followers_tweets* ← users.csv, tweets files for fake_followers
4: *genuine_accounts, genuine_accounts_tweets* ← users, tweets files for genuine_accounts
5: *all_users* ← **concatenate** *traditionalBots*, *socialBots*, *fake_followers*, *genuine_accounts*
6: **for** user in *all_users:*
7: **if** user[“geo_enabled”] **equals** NULL:
8: user[“geo_enabled”] ← 0
9: **if** user[“name”] **equals** NULL:
10: user[“name”] ← “”
11: **if** user[“description”] **equals** NULL:
12: user[“description”] ← “”
13: **for** tweet in traditionalBots_tweets, socialBots_tweets, fake_followers_tweets and genuine_accounts_tweets
14: **if** tweet[“text”] **equals** NULL:
15: tweet[“text”] ← “”

---

2 - Feature Extraction

After preprocessing was complete, the portion of the dataset we selected was ready for feature extraction. As the feature extraction approach, a hybrid approach was selected that contained features from both 3.2.1 and 3.2.2 while also adding some new ones. In total, 5 different feature groups, namely metadata, account-based, behavioral, content, and graph-based, for a total of 24 features were selected. The names, explanations, and pseudocode for each feature are given while formulas, given for appropriate features, and expected outcomes are provided for most of the features. Keep in mind that a feature that doesn’t have an expected outcome, doesn’t mean that it has no use. This means that for these types of features, we expect them to create relations not on their own but together as a whole to give us groupings that would help detect social bots. Also, “expect” here doesn’t mean that the feature results will always act according to the way they were designed, but it means that they’ll give enough of a majority to make our model work better.

2.a) Metadata Features

Metadata features are features that we can directly get from the user account attributes without doing any type of altercation on them. These features are important as they reflect the account in a very basic way. The metadata features selected were:

1 - Geo Enabled (GE): A boolean value that indicates whether a user-enabled geo location property is to be added to a tweet when publishing it.
2 - Statuses Count (SC): The total tweet count of the user.
3 - Favorites Count (FC): The total amount of tweets the user has liked.
4 - Friends Count (FRC): The total number of accounts this user is following.
5 - Followers Count (FOC): The total number of accounts that are following this user.

Pseudocode for Metadata Features:

1: *allUsersFeatures* ← **pandas** data frame
2: *allUsersFeatures*[“GE”] ← *all_users*[“geo_enabled”]
3: *allUsersFeatures*[“SC”] ← *all_users*[“statuses_count”]
4: *allUsersFeatures*[“FC”] ← *all_users*[“favourites_count”]
5: *allUsersFeatures*[“FRC”] ← *all_users*[“friends_count”]
6: *allUsersFeatures*[“FOC”] ← *all_users*[“followers_count”]

2.b) Account-Based Features

Account-based features are features that can be derived from the user account attributes. These features are more complex than the metadata features as some functional operations are being done on the attributes of the user. These features also reflect the account of the user in a more complex way. The account-based features selected were:

1 - Username Length (UL): The length of the account's username
2 - Screen Name Length (SNL): The length of the accounts screen name
3 - Description Length (DL): The length of the account's description
4 - Levenshtein Distance (LD): The Levenshtein distance between the accounts username and the screen name.
For Levenshtein distance, we expect the human account to be more creative than the bot accounts when creating their usernames and screen names. Because of this, we also expect the Levenshtein distance between the username and screen name of a human account to be higher in comparison to a bot account.
5 - Account Age (AA): The yearly age of the account.
Social bots are thought to be more lenient towards creating new accounts because of reasons such as suspension, getting blocked by other users, etc. On the flip side, as human users can use their accounts for a very long time without change, we expect human users to have higher account ages compared to bots.
6 - Tweets Count to Age Ratio (TCAR): The ratio of the total tweet count of the user to that user's account age.
As mentioned when discussing account age, humans are expected to have spent more time with an account. Adding to this, humans are also expected to have higher tweet counts since more lifetime of an account means more tweets in general. Bots are different in the sense that bots tend to go idle or stop using the account after some time which means they’ll have fewer tweets as the account ages. Because of this, we expect TCAR to go higher as the account ages increase for humans while it should go lower for bots as the account age increases.

$$TCAR = \frac{SC}{AA}$$

Pseudocode for Account-Based Features:

1: *allUsersFeatures*[“UL”] ← **length of** *all_users*[“name”]
2: *allUsersFeatures*[“SNL”] ← **length of** *all_users*[“screen_name”]
3: *allUsersFeatures*[“DL”] ← **length of** *all_users*[“description”]
4: *allUsersFeatures*[“LD”] ← **Levenshtein distance between** *all_users*[“name”] **and** *all_users*[“screen_name”]
5: **if** *all_users*[“created_at”].year **equals** DateTime.now.year:
6: *allUsersFeatures*[“AA”] ← 1 // Since AA will be needed for division
7: **else**
8: *allUsersFeatures*[“AA”] ← DateTime.now.year - *all_users*[“created_at”].year
9: *allUsersFeatures*[“TCAR”] ← *allUsersFeatures*[“SC”] / *allUsersFeatures*[“AA”]

---

2. c) Behavioral Features

Behavioral features are features that can be derived from the attributes of the tweets of the users. These features are the most complex as they try to analyze the certain patterns a bot might follow when tweeting.

1 - Tweet Time Mean (TTM): The mean of the hours in the day each tweet was tweeted.
Social bots are generally programmed to tweet in certain ways and times while humans tend to tweet more unpredictably. Because of this, we expect the TTM values of bots to be more grouped when compared to the TTM values of humans which should be more spread and unpredictable.

$$TTM = \frac{\sum_{i=0}^{k} creation\ hour\ of\ T_i}{SC}$$

2 - Tweet Time Standard Deviation(TTSD): The standard deviation of the hours in the day each tweet was tweeted.
TTSD is derived from TTM and also has the same idea behind it, with the idea being humans tweet more unpredictable than bots. Because of this, we expect the data of human accounts to be more spread compared to bots accounts, which in turn will make the standard deviation for human accounts much higher.

$$TTSD = \frac{\sum_{i=0}^{k} (creation\ hour\ of\ T_i - TTM)^2}{SC}$$

3 - Tweet Time Interval Mean (TTIM): The mean of the time differences between each tweet.
Following the idea that humans behave more unpredictably when tweeting, we devise another feature which is TTIM. This time, the time difference between each tweet is used instead of the hours each tweet was tweeted. Although the difference in the calculation, the expectation is similar as we still look for more grouped values for bot accounts and more spread out values for humans.

$$TTIM = \frac{\sum_{i=0}^{k} creation\ time\ of\ T_{i+1} - creation\ time\ of\ T_i}{SC}$$

4 - Tweet Time Interval Standard Deviation (TTISD): The standard deviation of the time differences between each tweet.
TTISD is derived from TTIM and also has the same approach as the other behavioral features covered so far, humans tweeting more unpredictably than bots. The same logic of more grouped values for bots and more spread out values for bots also apply and since the feature calculates standard deviation, more spread out means higher expected values for human accounts.

$$TTISD = \frac{\sum_{i=0}^{k} ((creation\ time\ of\ T_{i+1} - creation\ time\ of\ T_i) - TTIM)^2}{SC}$$

General Explanation for 5-6-7-8: Social bots use many different ways to spread the information they want. They use the help of retweets to spread misinformation, promote accounts; mentions to again promote accounts; hashtags to spread misinformation, go trending on Twitter with ads, and promotions; URLs to also spread misinformation, create malicious links, promote, and many more for more hostile activities. The tools Twitter provides for a regular user, retweets, mentions, hashtags, and URLs, can be used excessively by bots to achieve their goals. Because of this, the expectation is that RR, ART, AHT, and AUT will all be higher in bot accounts in comparison to human accounts.

5 - Retweet Ratio (RR): The ratio of the total retweeted tweet count of the user to the tweet count of the user.

$$RR = \frac{Total\ Retweeted\ Tweets\ Count}{SC}$$

6 - Average Mentions per Tweet (AMT): The total amount of mentions a user has per tweet.

$$AMT = \frac{Total\ Mentions\ Count}{SC}$$

7 - Average Hashtags per Tweet (AHT): The total amount of hashtags a user has per tweet.

$$AHT = \frac{Total\ Hashtag\ Count}{SC}$$

8 - Average URLs per Tweet (AUT): The total amount of URLs a user has per tweet.

$$AUT = \frac{Total\ URL\ Count}{SC}$$

Pseudocode for Behavioral Features:

```
//  It was mentioned before that tweet futures for each entity was computed separately
//  For the below code, assume that the tweets_of_entity is one of the tweet data frames and
user_entity is one of the user data frames
//  tweets_of_entity = one of genuine_accounts_tweets, fake_followers_tweets,
traditionalBots_tweets, socialBots_tweets
//  user_entity = one of genuine_accounts, fake_followers, traditionalBots, socialBots
//  Also, assume that they are used in the order they are given, one by one, to create
user_entity_tweet_features
// user_entity_tweet_features = one of genuine_accounts_tweet_features,
fake_followers_tweet_features, traditionalBots_tweet_features, socialBots_tweet_features

1: TTM ← 0
2: TTSD ← 0
3: TTIM ← 0
4: TTISD ← 0
5: totalMentions ← 0
6: totalURLs ← 0
7: totalHashtags ← 0
8: totalRetweetedTweets ← 0
9: totalTweetsCount ← 0
10: tweetTimesArray ← []
12: tweetHoursInDay ← []
12: tweetTimeIntervalsArray ← []
13: sample ← locate tweets where tweets_of_entity["user_id"] equals user_entities["id"]
14: for _, data in sample.iterrows:
15:     tweetText ← data["text"]
16:     totalTweetsCount ← totalTweetsCount + 1
18:     totalMentions ← totalMentions + data["num_mentions"]
18:     totalURLs ← totalURLs + data["num_urls"]
19:     totalHashtags ← totalHashtags + data["num_hashtags"]
20:     if tweetText[0:4] equals "RT @":
21:         totalRetweetedTweets ← totalRetweetedTweets + 1
22:     append data["created_at"] to tweetTimesArray
23:     //  Design choice to not waste O(n) computational time for array operations
24:     //  Users with less than 4 tweets will be filtered anyways
25:     //  This filtering will be discussed in post-processing
26:     if 4 <= length of tweetTimesArray: //  Design choice
27:         firstTweetTime ← tweetTimesArray[0]
28:         append firstTweetTime.hour to tweetHoursInDay
29:         for i in range length of tweetTimesArray:
30:             traverseTweetTime ← tweetTimesArray[i]
31:             append traverseTweetTime.hour to tweetHoursInDay
32:             timeDifferenceBetweenTweets ← daily time difference between
firstTweetTime and traverseTweetTime
33:             append timeDifferenceBetweenTweets to tweetTimeIntervalsArray
34:             firstTweetTime = traverseTweetTime
35:         TTM ← mean of tweetHoursInDay
36:         TTSD ← standard deviation of tweetHoursInDay
```

```
37:             TTIM ← mean of tweetTimeIntervalsArray
38:             TTISD ← standard deviation of tweetTimeIntervalsArray
39:     user_entity_tweet_features["TTM"] ← TTIM
40:     user_entity_tweet_features["TTSD"]  ← TTSD
41:     user_entity_tweet_features["TTIM"] ← TTIM
42:     user_entity_tweet_features["TTISD"] ← TTISD
43:
44:     //  O(1) computations are ignored in cases of filtering
45:     if totalTweetsCount equals 0:
46:             user_entity_tweet_features["RR"] ← 0
47:             user_entity_tweet_features["AMT"] ← 0
48:             user_entity_tweet_features["AHT"] ← 0
49:             user_entity_tweet_features["AUT"] ← 0
50:     else
51:             user_entity_tweet_features["RR"] ← totalRetweetedTweets /
totalTweetsCount
52:             user_entity_tweet_features["AMT"] ← totalMentions / totalTweetsCount
53:             user_entity_tweet_features["AHT"] ← totalHashtags / totalTweetsCount
54:             user_entity_tweet_features["AUT"] ← totalURLs / totalTweetsCount
```

2.d) Content Features: Content features are features that can be derived from the attributes of the tweets of the users. These features can also be thought of as features that can be derived from the publicly available content of a tweet.

1 - Average Tweet Size (ATS): The average length of a tweet.
As discussed before, bots tend to exploit Twitter tools such as mentions, URLs, hashtags, etc. This causes the tweets of the bot accounts to be longer than of the human accounts so for ATS, we expect it to be higher for bot accounts in comparison to human accounts.

$$ATS = \frac{\sum_{i=0}^{k} size\ of\ T_i}{SC}$$

General Explanation for 2-3: Since most of the followers of bot accounts are also bots, this means that the social interaction from the bot's followers will be minimal. This will result in the tweets of bots not getting many likes or retweets. Because of this, we expect the ARF and AFT features to be lower for bots in comparison to humans.

2 - Average Retweets per Tweet (ART): The average retweet count a tweet has per tweet.

$$ART = \frac{\sum_{i=0}^{k} retweet\ count\ of\ T_i}{SC}$$

3 - Average Favorites per Tweet (AFT): The average favorite count a tweet has per tweet.

$$AFT = \frac{\sum_{i=0}^{k} favorite\ count\ of\ T_i}{SC}$$

Pseudocode for Content Features:

```
//  Continuing from the loop in behavioral features, assume this code is written in the for loop
//  Loop start:
//  sample ← locate tweets where tweets_of_entity["user_id"] equals user_entities["id"]
//   for _, data in sample.iterrows:

totalLikesToTweetsCount ← 0  //  Assume variable is declared before the loop
totalRetweetsToTweetsCount ← 0  //  Assume variable is declared before the loop
totalTweetSize ← 0

//  Continuing in the loop from here on out
for _, data in sample.iterrows:  //  Demonstration of loop-start
    totalLikesToTweetsCount ← totalLikesToTweetsCount + data["favourite_count"]
    totalRetweetsToTweetsCount ← totalRetweetsToTweetsCount + data["retweet_count"]
    totalTweetSize ← totalTweetSize + length of tweetText

    if totalTweetsCount equals 0:
        user_entity_tweet_features["ATS"] ← 0
        user_entity_tweet_features["ART"] ← 0
        user_entity_tweet_features["AFT"] ← 0
    else
        user_entity_tweet_features["ATS"] ← totalTweetSize / totalTweetsCount
        user_entity_tweet_features["ART"] ← totalRetweetsToTweetsCount / totalTweetsCount
        user_entity_tweet_features["AFT"] ← totalLikesToTweetsCount / totalTweetsCount
```

2.e) Graph-Based Features: Graph-based features are features that can be derived from social graphs of Twitter users. Twitter allows each users to build a social graph for themselves where they can follow or get followed by other users. Features extracted from this social graph are called graph-based features.

General Explanation for 1-2: Social bots try to gain followers by following many random accounts, hoping for a follow back. This creates a scenario where the friend count of the bot is inflated while the follower count remains low, resulting in FFR and RS values being lower in bot accounts in comparison to human accounts.

1 - Follower to Friends Ratio (FFR): The ratio of the total follower count of the user to the total count of accounts the said user follows.

$$FFR = \frac{FOC}{FRC}$$

2 - Reputation Score (RS): The ratio of the total follower count of the user to the sum of the total count of accounts the said user follows and the users' follower count.

$$RS = \frac{FOC}{FOC + FRC}$$

Pseudocode for Graph-Based Features:

1: **if** *allUsersFeatures*[“FRC”] **equals** 0:
2: *allUsersFeatures*[“FFR”] ← 0 // Since zero by division error
3: **else**
4: *allUsersFeatures*[“FFR”] ← *allUsersFeatures*[“FOC”] / *allUsersFeatures*[“FRC”]
5: **if** *allUsersFeatures*[“FRC”] **equals** 0:
6: *allUsersFeatures*[“RS”] ← 0 // This is a design choice
7: **else**
8: *allUsersFeatures*[“RS”] ← *allUsersFeatures*[“FOC”] / (*allUsersFeatures*[“FOC”] + *allUsersFeatures*[“FRC”])

All of the features selected for our model can be seen in Figure 3.5. Keep in mind that all metadata, account-based, and graph-based features are user features since they can be derived from user attributes while behavioral and content features are all tweet features since they can be derived from tweet attributes.

Category	Description	Feature
Metadata	Geo Enabled (GE)	(1)
	Statuses Count (SC)	(2)
	Favourites Count (FC)	(3)
	Friends Count (FRC)	(4)
	Followers Count (FOC)	(5)
Account-Based	Username Length (UL)	(6)
	Screen Name Length (SNL)	(7)
	Description Length (DL)	(8)
	Levenshtein Distance (LD)	(9)
	Account Age (AA)	(10)
	Tweets Count to Age Ratio (TCAR)	(11)
Behavioral	Tweet Time Mean (TTM)	(12)
	Tweet Time Standard Deviation (TTSD)	(13)
	Tweet Time Interval Mean (TTIM)	(14)
	Tweet Time Interval Standard Deviation (TTISD)	(15)
	Retweet Ratio (RR)	(16)
	Average Mentions per Tweet (AMT)	(17)
	Average Hashtags per Tweet (AHT)	(18)
	Average URLs per Tweet (AUT)	(19)
Content	Average Tweet Size (ATS)	(20)
	Average Retweets per Tweet (ART)	(21)
	Average Favourites per Tweet (AFT)	(22)
Graph-Based	Follower to Friends Ratio (FFR)	(23)
	Reputation Score (RS)	(24)

Figure 3.5: Features of Our Model

The feature extraction was done on the user and tweet features separately to then be concatenated. For the user features, we combined all the user CSV files on top of each other to get a final CSV file that contained all 11282 user accounts. Since this file wasn’t

too big, we were able to extract useful features without having any trouble with computation time. We also kept the IDs of the users and labeled them. For the tweet features, since the data we were dealing with was huge and we were using O(n) complexity calculations such as array loops, doing each feature extraction one by one took less time than if we had concatenated all tweets together. For human accounts and each type of bot account, feature extraction was done one by one. We were able to do this type of feature extraction by using the foreign key relation discussed in Figure 3.2. Each tweet had an attribute “user_id”, giving us information on which user sent the tweet. By going through all the users one by one and sampling all their tweets to that user's id, we were able to extract the tweet features. Since this type of extraction was done four times, one for human accounts and a total of three times for each bot type, we made use of the IDs given in user attributes for each time we were doing feature extraction. Ids were also kept in the final files. At the end of the feature extraction, we had a total of five CSV files, one file that contained all 11282 users id’s, their user features; four other files that contained id’s the specific types of accounts, human, social, traditional bot, fake follower, and their tweet features.

Post-Processing

After the feature extraction was done, we were left with one file that contained all the users’ IDs, their user features, and four files that contained each type of accounts id’s, and their tweet features. Now, we needed to combine these features and get them ready for the testing part. Firstly, we concatenated all the tweet feature files into a single file, which gave us a CSV file with an equal row count to the one that has the user features. Remember that we had also kept the ids of the users in both the user and tweet feature files. Using these IDs, we put these files together to create a final CSV file that contained all the users, a total of 11282 users, we’ve selected and their user, tweet features with labels for each user. One last thing we did was to eliminate users that had a tweet count of three or less. We thought this would be appropriate as eleven of the features we have, nearly half of the features, are tweet features. This constitutes a big portion of the feature data so we tried to eliminate users that might cause noisy data. In the end, we ended with a total of 8071 users, 1082 human, and 6989 bot accounts. The reason so many users were filtered was because of the computational overhead the python libraries had when dealing with huge memory requirements, which was discussed in section 1.1. On the other hand, bot accounts were fine and they only got filtered because many bot accounts had little amounts of tweets. The final filtered CSV file that contained labels and user and tweet features was then tested with various machine learning models to find the one that gave the best accuracy. The model with the best accuracy would also be the model that we would use on our website. We’re not going into much detail about the testing and evaluation of the features as they’ll be more deeply discussed in sections 4 and 5.

Pseudocode for Post-Processing:

// *user_entity_tweet_features* = **one of** *genuine_accounts_tweet_features, fake_followers_tweet_features, traditionalBots_tweet_features, socialBots_tweet_features*

1: *allTweetFeatures* ← **concatenate all** *user_entity_tweet_features*
2: *allFeatures* ← **merge** allTweetFeatures **with** *allUsersFeatures* **by** id
3: *allFeatures* ← *allFeatures*[*allFeatures*[“SC”] >= 4]

3.3 Dynamic Model

Since the main part of our project is creating a machine learning model and this makes the project software-intensive, our dynamic model is very simple. We have one very simple use case which is what we train our machine learning model for, checking if a Twitter account is a social bot. Let's call this use case "Check Prediction". In Figure 3.6, our use case diagram can be seen. The diagram is, as mentioned, very simple and contains the essential use case of our project. Also, in Table 3.1, a detailed overview of our essential use case can be seen.

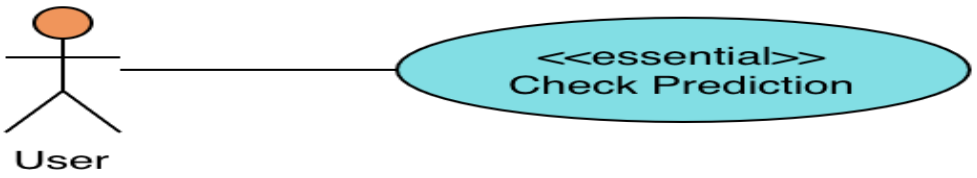


Figure 3.6: Use Case Diagram of our Design

Table 3.1: Overview of the Use Case "Check Prediction"

Use-Case Name	Check Prediction
Purpose	To understand if a Twitter account is a bot, the user must check the models' prediction
Overview	1 - A user enters the website. 2 - The system asks for the Twitter username of the account to be searched. 3 - The user enters the username of the Twitter account. 4 - The system checks if the Twitter account exists. 5 - The system checks if the Twitter account isn't suspended. 6 - The system makes the prediction. 7 - The system displays the prediction for the user to see.
Actor	User
Preconditions	1 - The user must have an internet connection while accessing the website. 2 - The user must know the username of the Twitter account to be searched. 3 - The Twitter account must exist for the user to see a prediction.
Exceptions	4.a - The Twitter account doesn't exist: The user is shown an error message displaying the error. The use case ends. 5.a - The Twitter account is suspended: The user is shown an error message displaying the error. The use case ends.
Postcondition	1 - A result is displayed, either the prediction or an error message.

The essential use case of our project “Check Prediction”, gives an overview of the steps both the user and the system go through. The user enters the website to check the prediction of our model for the Twitter account they are interested in. Then, the system shows them a screen where the username is prompted to enter. The user enters the username and here, the system takes over the user's actions. The system verifies the Twitter account by checking error conditions, account existence, and its suspension status in order. Lastly, depending on if the account was verified, the system directly displays an error message or it makes a prediction and then displays this prediction. In both cases, a result is displayed to the user. For further illustration of the use case, the sequence diagram given in Figure 3.7 can also be checked.

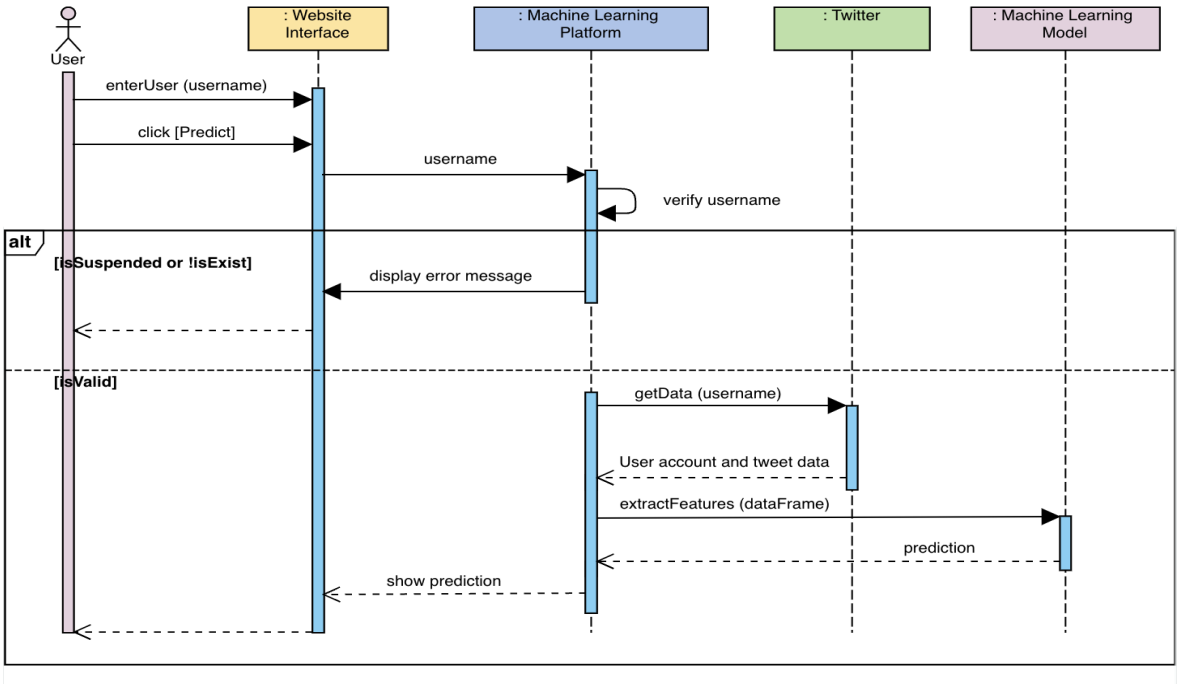


Figure 3.7: Sequence Diagram of our Design

Complementing our essential use case, the interface of our website was designed to reflect its simplicity. The website interface is a two-page design using a navigation bar where we include the main page, where the user can enter the username of the Twitter account to be searched, and an FAQ page, where we discuss some matters that might be important to the user. Since, as mentioned before, the dynamic model of our system is simple; our whole interface can be checked in Figures 3.8, 3.9, and 3.10 without the need for mock-ups. The first page of the website is the main page, shown in Figure 3.8, and it’s designed to be straight to the point as it prompts the user to enter the username of the Twitter account to be searched for. An example is given to the user in which the expected format is shown. Also, the name of our application, Twitter Bot Detector, and a bot emoji are shown to express the main goal/subject of our system to the user. The second page of the website is the FAQ page, shown in Figures 3.9 and 3.10, and it is designed to express some important topics such as the definition of a bot, how the system works, and how the results should be interpreted. There’s also a section where privacy matters regarding the data are discussed with a link to this report which is uploaded to GitHub. With this, we tried to be as transparent as possible with the data usage and our approach when creating the model to relieve any privacy concerns that a user might have.

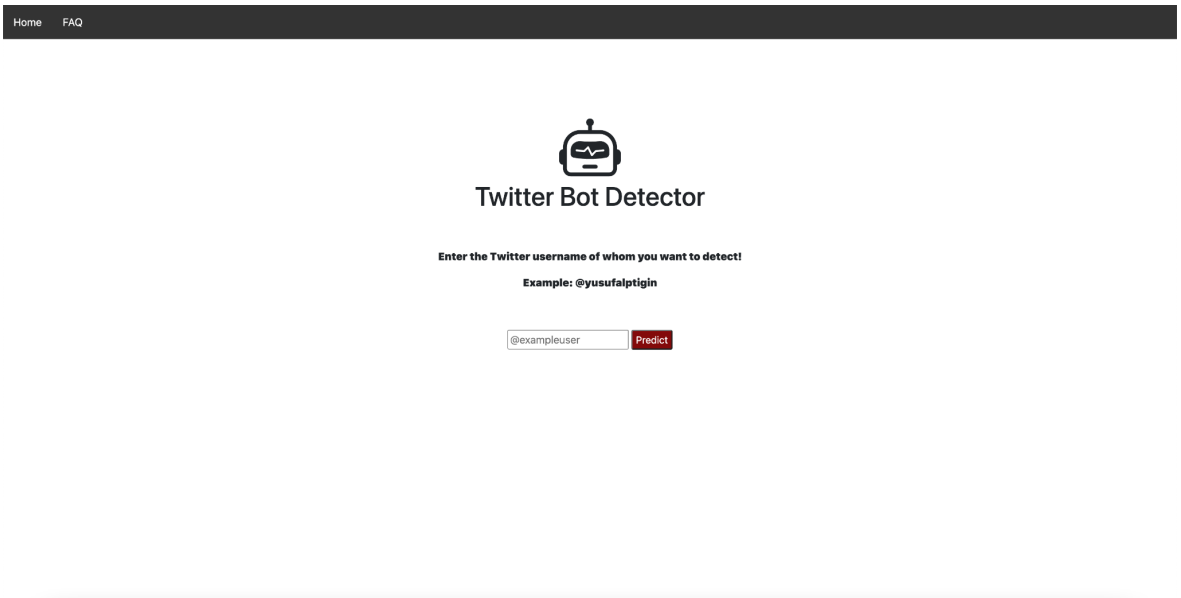


Figure 3.8: Home Page

Figure 3.9: FAQ Page

Figure 3.10: FAQ Page Continued

4 Experimentation Environment and Experiment Design

In this section, we discuss the types of experiments we've built to measure the different aspects of our designs that were discussed in 3.2.1, 3.2.2, and 3.3.3. To find the best possible model to use on our website, we create a basis for our experiments to then be able to do live testing on the selected models.

4.1. Evaluation Metrics and Selected Classifiers

To have a basis for our experiments, we chose 8 standard evaluation metrics with 6 different classifiers and tested all our designs for these metrics. Namely, the standard metrics chosen were accuracy, precision, error rate, recall, specificity, f1-Score, miss rate, and false-positive rate while the machine learning classifiers were chosen as, Random Forest, AdaBoost, XGBoost, Logistic Regression, Naive-Bayes, k-NN. Choosing a broader approach of both evaluation metrics and classifiers, we wanted to have many different options on which model to choose to put on our website. Also, creating such a basis for our experiments we wanted to have a "two birds in one stone" type of effect as the results from these experiments would allow us to both compare the results of our work to existing literature while also giving us the ability to compare the models in themselves to pick the best model possible to go live with. During the experiments, the idea would be to, for the classifier that is being tested, create a confusion matrix, to then calculate the necessary evaluation metrics from it. In Table 4.1, a sample confusion matrix is given. Also, all 8 of the evaluation metrics chosen for are experiments and how to calculate them from the confusion matrix can be seen below in Table 4.1. For the confusion matrix, remember that we've labeled bots as 1's and humans as 0's.

Table 4.1: A Sample Confusion Matrix

	Actually Negative	Actually Positive
Predicted Negative (0)	True Negatives (TNs)	False Negative (FNs)
Predicted Positive (1)	False Positive (FPs)	True Positives (TPs)

$$Accuracy\ (ACC) = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Error\ Rate\ (ERR) = \frac{FP + FN}{TP + FP + TN + FN}$$

$$Precision\ (PRE) = \frac{TP}{TP + FP}$$

$$Recall\ (REC) = \frac{TP}{TP + FN}$$

$$Specificity\ (SPE) = \frac{TN}{TN + FP}$$

$$f1 - Score\ (F1S) = \frac{2\,.\,precision\,.\,recall}{precision + recall} = \frac{2\,.\,TP}{2\,.\,TP + FP + FN}$$

$$Miss\ Rate - False\ Negative\ Rate\ (FNR) = \frac{FN}{FN + TP}$$

$$False\ Positive\ Rate\ (FPR) = \frac{FP}{FP + TN}$$

4.2 Choosing the Model and Data Collection

After doing all the experiments discussed, we had to choose a model to put on our website that would be used live when a user wants to check a prediction for a live Twitter account. The experimentation conclusion and which model we picked, hint: It was a random forest, is discussed in Section 5. To choose the best model that we would work on live Twitter accounts, we had to do live testing on the model that gave the best performance. To do the live testing, we created a small database that consisted of 50 human and 50 bot accounts. These accounts were specifically and randomly hand-picked by us with the help of the online publicly available bot repository [32]. In this repository, datasets of humans and bots exist with their Twitter IDs and labels.

For the user accounts, 50 of them were picked from either the bot repository or from people that we knew for certain were bots. The 50 accounts also had different types of human accounts such as sophisticated users, users that have millions of followers, verified accounts, accounts that are verified but don't have enough followers to be classified as sophisticated accounts, and regular users like your average Twitter user. While most of the sophisticated and verified users were hand-picked from the bot repository, all of the regular users were picked by us. The regular users consisted of our friends, family, friends of our friends, and people we can verify from youtube twitch, etc., so in general, accounts we knew with absolute certainty that they were bots.

For the bot accounts, we resorted to mostly picking them from the bot repository. Since the repository consisted of different types of bots, we were able to make a mix of bot types to live test our model. Even though it was out of our topic, we also included a few bots that weren't malicious to see how our model would do. Unfortunately, we weren't able to add political bots to our live dataset as when we checked around 30-40 accounts and saw that all were suspended, we stopped picking further. For a small portion of the live bot dataset, we also added hand-picked bots too. This was done by going through Twitter accounts that followed our accounts and finding the bot ones and their followers, and friends. Just to make sure these accounts were actual bots, we were picking by eye at this point, we verified our picks with other bot detection strategies such as BotoMeter [33].

5 Comparative Evaluation and Discussion

In this section, we analyze the three approaches we've applied for bot detection, Language-Agnostics, One-Classification, and Novel Hybrid, using the basis we've created for our experiments as the main stepping stone. The basis experiment was done for all the approaches, but as the Language-Agnostic and One-Classification approaches were merely do-overs, we didn't do further experiments. Instead, a deeper, exhaustive discussion and evaluation is given for our novel hybrid approach, where we discuss the basis experiment results, comparison of our approach with approaches in the literature, live testing results, model selection, and goals, evaluation criteria given in our interim report.

5.1 Language-Agnostic Approach

In table 5.1, our results for our basis experiment, 6 machine learning classifiers, and 8 different evaluation matrices, can be seen. These results were achieved using a combination of both user and tweet features of accounts. From these results, we can see that XGBoost, AdaBoost, and Random Forest give the best results with accuracies of 0.9907, 0.9885, and 0.9914. After these 3, we have k-NN which still has a high accuracy of 0.9782 but still, it's not as good as the first three classifiers mentioned. After these two, we have Logistic Regression and Naive-Bayes algorithms with accuracies of 0.9572 and 0.9103. The trend where the first three classifiers are the best followed by k-NN, Logistic Regression, and Naive-Bayes will be a common trend as it'll be seen further in the discussion. The order of the algorithms from the best accuracy to worst is Random Forest > XGBoost > AdaBoost > k-NN > Logistic Regression > Naive-Bayes.

Table 5.1: Language-Agnostic Approach Results

	ACC	ERR	PRE	REC	SPE	F1-S	FNR	FPR
XGBoost	0.9907	0.0092	0.9874	0.9788	0.9952	0.9831	0.0211	0.0080
AdaBoost	0.9885	0.0114	0.9796	0.9786	0.9923	0.9791	0.0213	0.0080
Random Forest	0.9914	0.0085	0.9890	0.9890	0.9958	0.9844	0.0200	0.0075
Logistic Regression	0.9572	0.0439	0.8530	0.9851	0.9469	0.9143	0.0148	0.0048
Naive-Bayes	0.9103	0.0896	0.8933	0.8010	0.9581	0.8446	0.1989	0.0832
k-NN	0.9782	0.0217	0.9636	0.9570	0.9863	0.9603	0.0429	0.0162

Looking at the precision values, we can also see the first 4 algorithms, accuracy-wise, don't have a high cost for predicting positives as the precision values rise up in parallel with the accuracy values. In other words, these 4 algorithms do well when they predict an account to be a bot. In addition to this, the recall values of these 4 algorithms also show that they can detect the bot accounts at a similar rate as they detect all accounts. These values major significantly in the other two approaches though. As for Naive-Bayes, even

though the precision value is quite similar to its accuracy, the cost of finding bots isn't that high, it's only able to detect bots at a rate of 0.8. This is 0.1 less than its accuracy. For Logistic Regression, we can see that it's able to detect bots at a rate of 0.9851, which is a lot higher than its accuracy and close to the total accuracy of the 3 best classifiers, but its cost of detecting these bots, precision value is low, is quite high.

5.2 One-Classification Approach

Just as we've done in the previous approach, we've tested our results with the basis experiment we've created. Experimentation results for this approach can be seen in Table 5.2. The results went quite similar to the results in 5.1, as the order of the algorithms from the best accuracy to worst was Random Forest > XGBoost > AdaBoost > k-NN > Logistic Regression > Naive-Bayes. The precision and recall values were also quite similar in the 4 best, accuracy-wise, classifiers. The k-NN value for precision was just a little lower in ratio when compared to the language-agnostic approach, meaning it is a little more expensive to detect bots but that's about it. Logistic Regression is also quite similar as it's able to detect most of the bots, even coming close to the total accuracies of the 3 best algorithms. But its precision value is significantly less, meaning it's a whole lot more expensive to detect bots. The main difference in this approach though comes in the Naive-Bayes algorithm as the recall value of it is 0.9898, meaning it detects nearly all bots. This comes at a price though as the precision value of it's extremely low at 0.5141, meaning it's super expensive to detect the bots, even though it's able to detect nearly all of them. The best results from the two approaches can be seen in Table 5.3 and it shows that when the best-resulting models are picked, the differences aren't so apparent.

Table 5.2: One-Classification Approach Results

	ACC	ERR	PRE	REC	SPE	F1-S	FNR	FPR
XGBoost	0.9899	0.0100	0.9849	0.9782	0.9943	0.9816	0.0217	0.0081
AdaBoost	0.9897	0.0102	0.9849	0.9782	0.9943	0.9816	0.0217	0.0081
Random Forest	0.9900	0.0099	0.9841	0.9796	0.9940	0.9818	0.0203	0.0076
Logistic Regression	0.9322	0.0667	0.7752	0.9746	0.9218	0.8635	0.0253	0.0075
Naive-Bayes	0.8660	0.1339	0.5141	0.9898	0.8455	0.6767	0.0101	0.0019
k-NN	0.9684	0.0315	0.9110	0.9714	0.9873	0.9402	0.0285	0.0100

Table 5.3: Best Results from Both Approaches

	ACC	ERR	PRE	REC	SPE	F1-S	FNR	FPR
5.1 (RF)	0.9914	0.0085	0.9890	0.9890	0.9958	0.9844	0.0200	0.0075
5.2 (RF)	0.9900	0.0099	0.9841	0.9796	0.9940	0.9818	0.0203	0.0076

5.3 Novel Hybrid Approach

For our novel hybrid approach, in addition to doing the basis experiments we had in the other two approaches, we delved deeper into other experimentations, visualizations, and discussions. We visualized our results, compared our approach with other approaches in the literature, did live testing and model selection, and discussed our final system by evaluating the goals and evaluation criteria we had in our interim report.

5.3.1 Basis Testing Results

For our novel hybrid approach, we again used our testing basis of 8 evaluation metrics and 6 classification models. A little different from the testing of the previous two approaches, for the testing part, we went with a 20-fold cross-validation approach. The reason we didn't use the generally chosen 10-fold cross-validation was because of the constraints the dataset put on us. After the post-processing, we were left with a total of 8071 users, 1082 human, and 6989 bot accounts. If we had used 10-fold cross-validation on this dataset, for every iteration, there would be around 807 users for testing. Seeing that we had a total of 1082 human accounts and at most 1000 traditional spambots, some might be filtered after post-processing, and 10-fold cross-validation had a serious chance of wiping out most of a user type. This in the end didn't make sense as we were trying to create a generalized model and also, testing our model on types of users that it's not trained on wasn't intuitive. Because of these, we went with a 20-fold cross-validation approach. In Table 5.4, the results of our novel model with 6 classification models for 8 different evaluation metrics using 20-fold cross-validation. Graphic representations of each evaluation metric for all 6 classifiers can also be seen in Figures 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, and 5.8.

Table 5.4: Novel Hybrid Approach Results

	ACC	ERR	PRE	REC	SPE	F1-S	FNR	FPR
XGBoost	0.9946	0.0053	0.9959	0.9978	0.9744	0.9969	0.0021	0.0255
AdaBoost	0.9934	0.0065	0.9954	0.9969	0.9707	0.9962	0.0030	0.0292
Random Forest	0.9936	0.0063	0.9957	0.9969	0.9725	0.9963	0.0030	0.0275
Logistic Regression	0.9762	0.0237	0.9878	0.9847	0.9198	0.9862	0.0152	0.0801
Naive-Bayes	0.9575	0.0424	0.9855	0.9660	0.8926	0.9757	0.0339	0.1073
k-NN	0.9824	0.0175	0.9885	0.9911	0.9272	0.9898	0.0088	0.0727

The results from Table 5.4 show that our novel hybrid model does really well as we're almost approaching a perfect accuracy of 1 in three types of classifiers, XGBoost, AdaBoost and Random Forest. Even though these accuracy results are not as high in the other 3 classifiers, Logistic Regression, Naive-Bayes, and k-NN, they are still in the very acceptable range and they give us really good results.

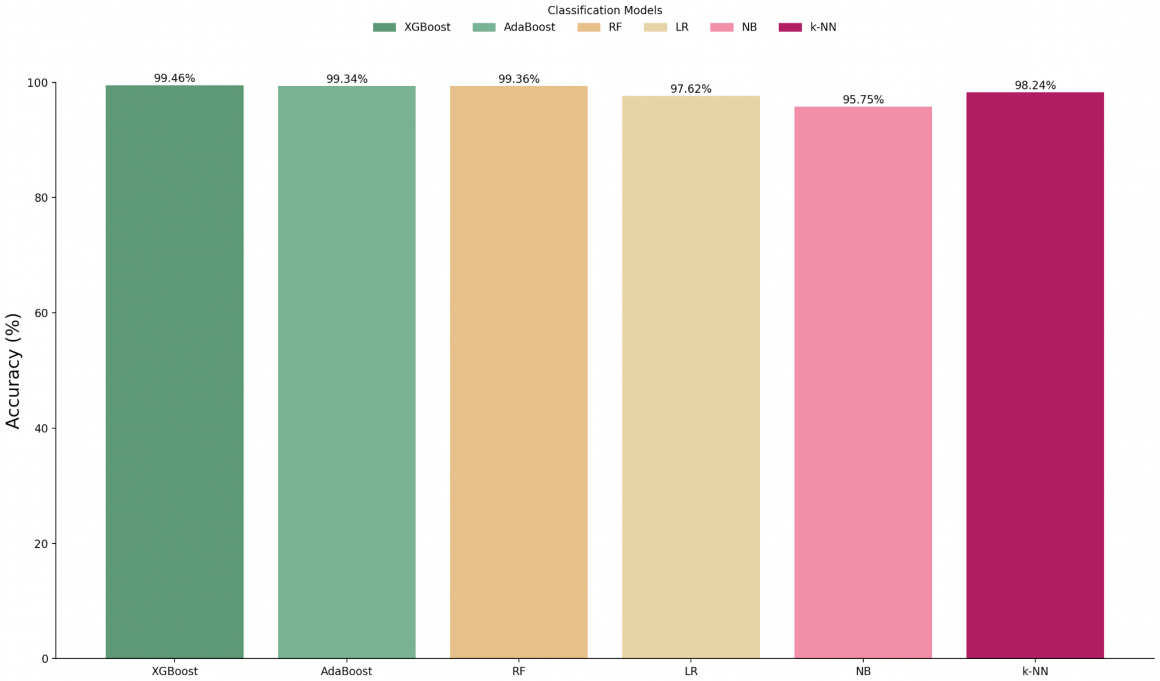


Figure 5.1: Accuracy of all Classifiers Given in %

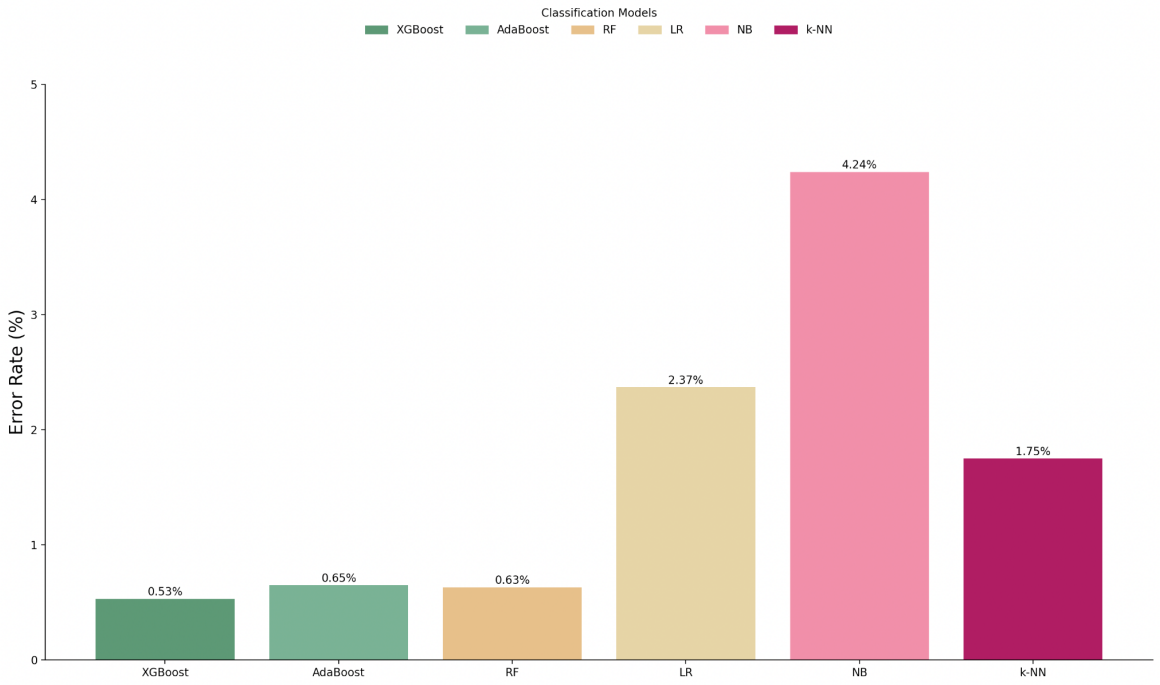


Figure 5.2: Error Rate of all Classifiers Given in %

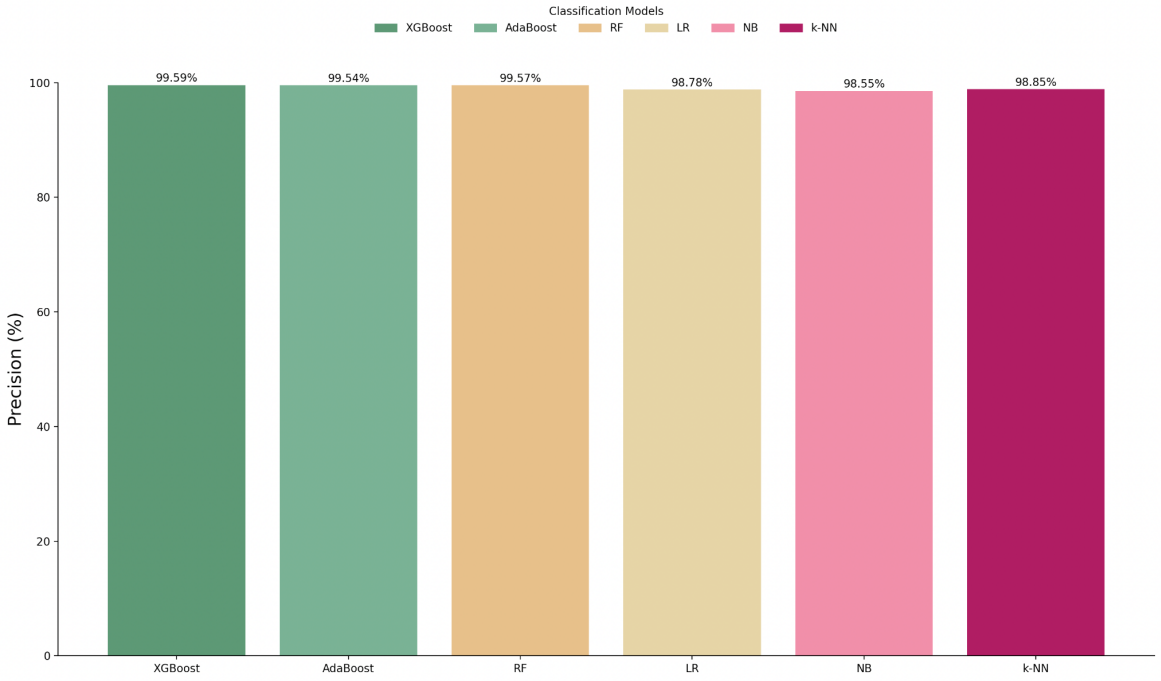


Figure 5.3: Precision of all Classifiers Given in %

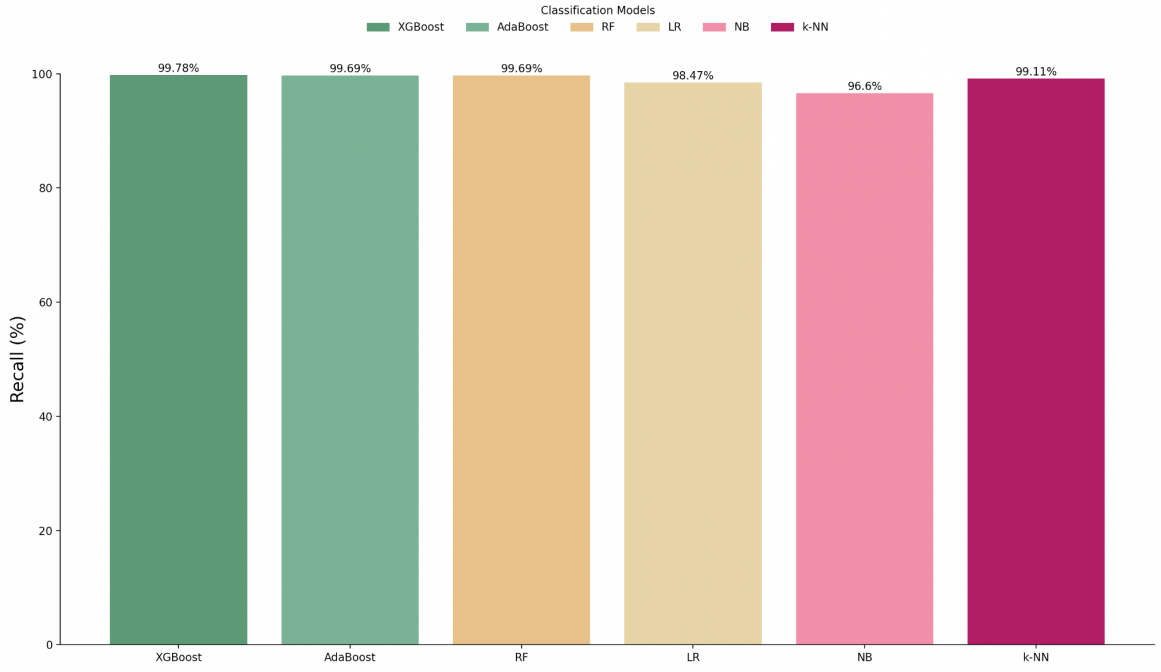


Figure 5.4: Recall of all Classifiers Given in %

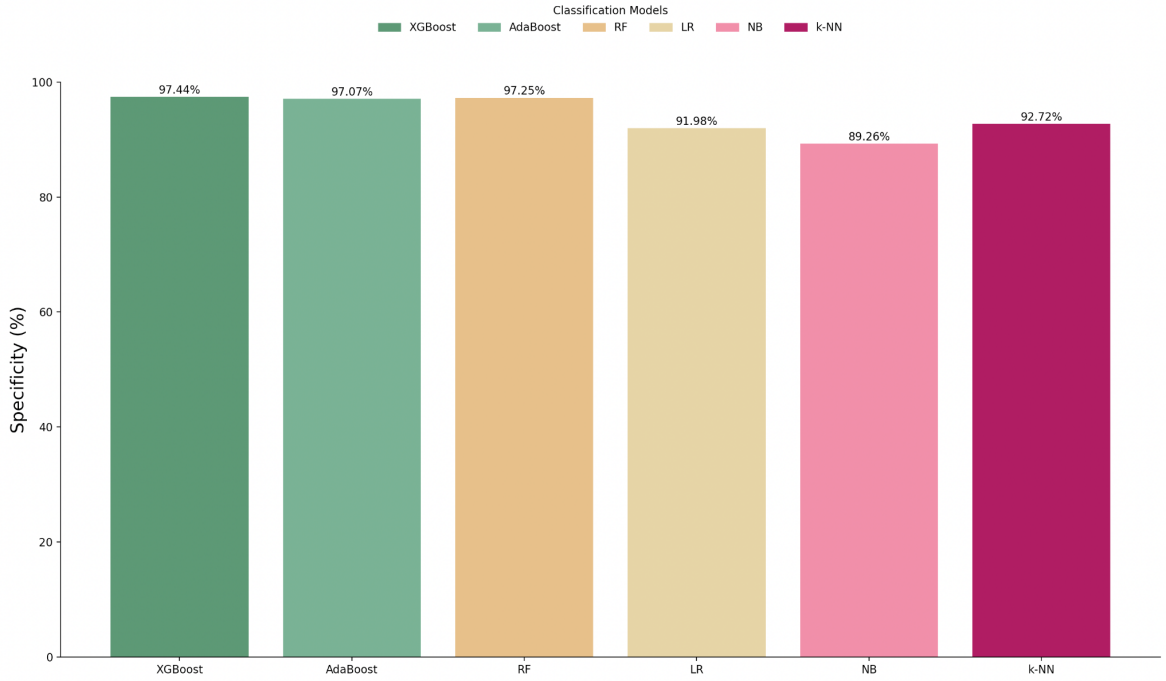


Figure 5.5: Specificity of all Classifiers Given in %

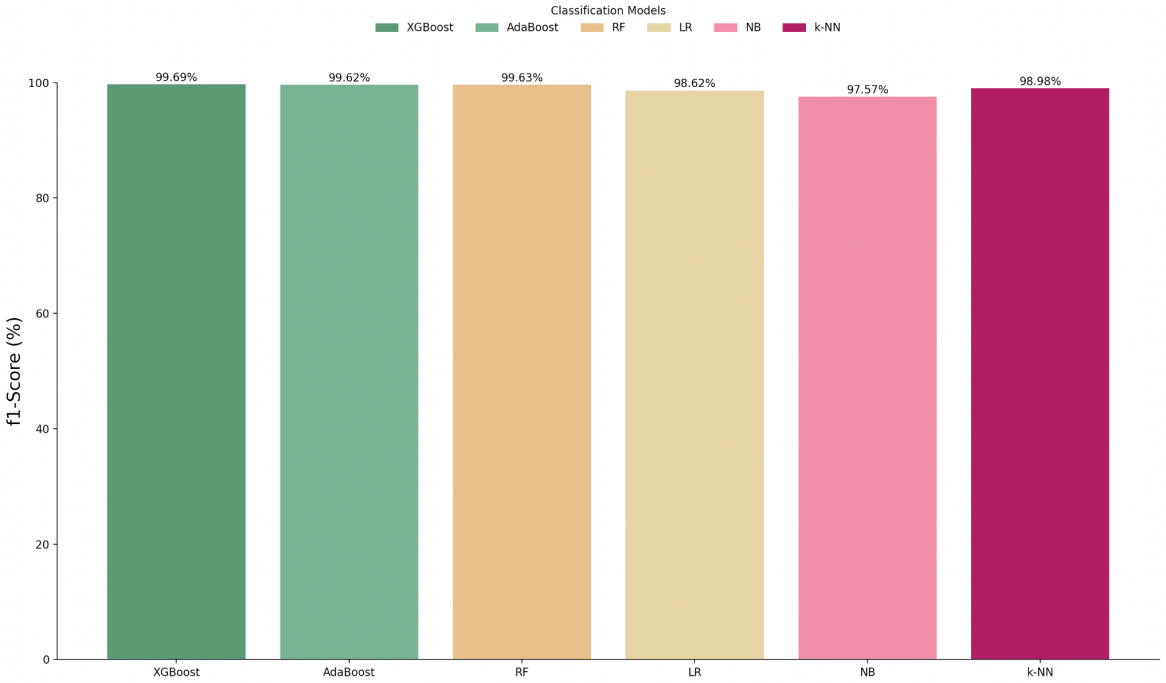


Figure 5.6: f1-Score of all Classifiers Given in %

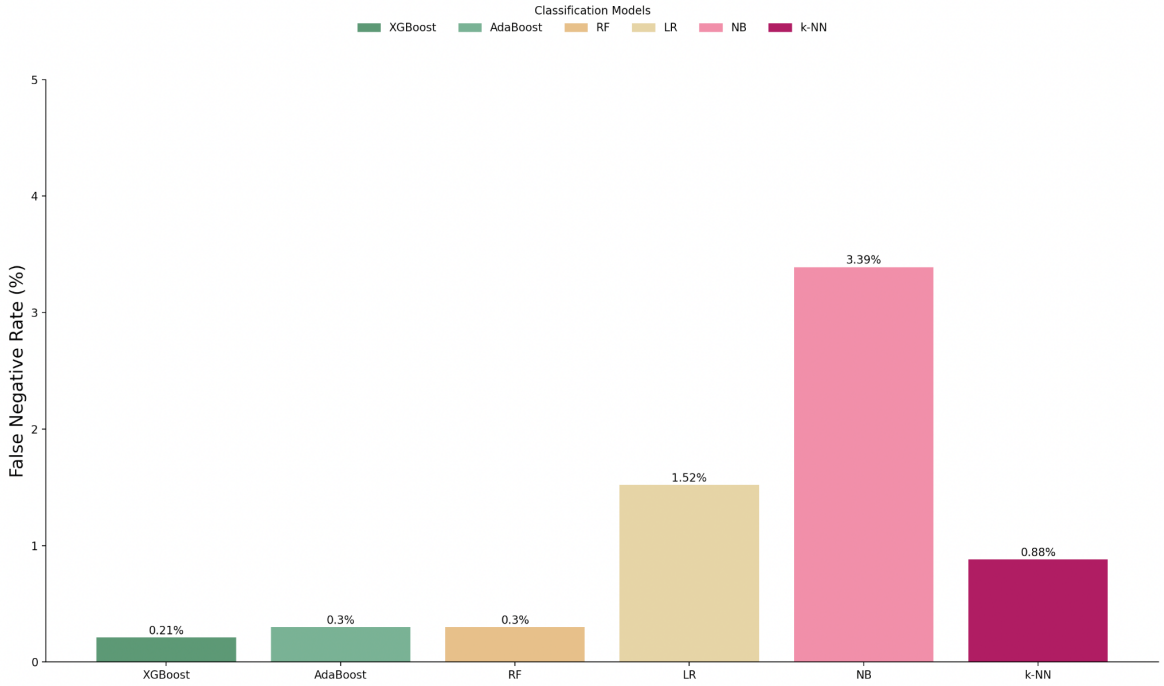


Figure 5.7: False Negative Rate of all Classifiers Given in %

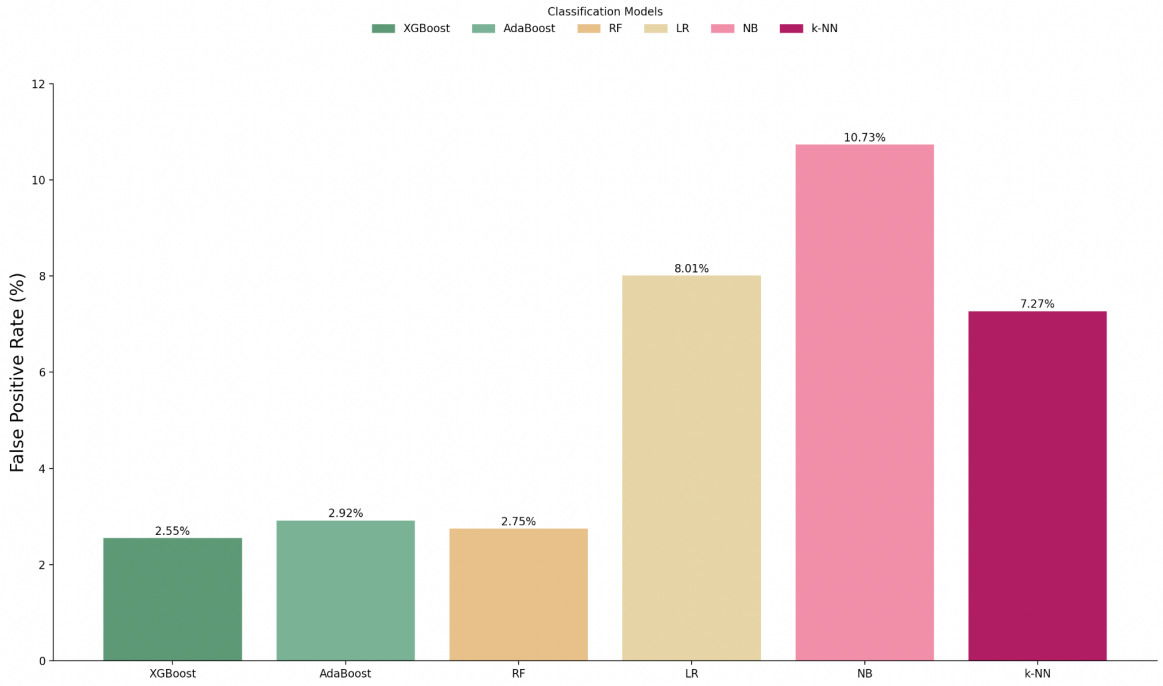


Figure 5.8: False Negative Rate of all Classifiers Given in %

5.3.2 Comparison with Approaches in Literature

To compare our results with approaches in the literature, we'll be using the two approaches we've sought out, [18] and [31], that also use the Cresci-17 dataset. In [18], J. Knauth was able to achieve accuracy, f1-Score, recall, and precision of 0.9881, 0.9896, 0.9835, and 0.9959. These results were achieved by their best-performing classifier AdaBoost. Comparing our results using AdaBoost to theirs, we can see that our model has done much better than them. We achieve an accuracy of 0.9934 while their approach only achieves 0.9881. Also, both our f1-Score and recall surpass theirs too as we have an f1-Score of 0.9962 and a recall score of 0.9969 while they have 0.9896 and 0.9835. The only evaluation metric that surpasses ours is precision as they've achieved a precision of 0.9958 while we've achieved 0.9954. Still, the difference is only 0.0004. In addition to this, this comparison was done by comparing our AdaBoost classifiers to theirs. While the AdaBoost classifier is their best performing one, we have two classifiers, Random Forest and XGBoost, that do better than our AdaBoost classifier. Our second best classifier, Random Forest, passes their AdaBoost in accuracy, f1-Score, recall while failing just a little bit by 0.0001 on precision. Our best performing classifier, AdaBoost, can pass their AdaBoost on all evaluation metrics, accuracy, f1-Score, recall, and precision. In Tables 5.5, 5.6, and 5.7; the comparison of our results to J. Knauth [18] can be seen.

Table 5.5: Comparison of our Novel Hybrid Approach with J. Knauth, AdaBoost Classifier

	ACC	F1-S	PRE	REC
J. Knauth (AdaBoost)	0.9881	0.9896	**0.9958**	0.9835
Our Approach (AdaBoost)	**0.9934**	**0.9962**	0.9954	**0.9969**

Table 5.6: Comparison of our Novel Hybrid Approach with J. Knauth, Random Forest Classifier

	ACC	F1-S	PRE	REC
J. Knauth (AdaBoost)	0.9881	0.9896	**0.9958**	0.9835
Our Approach (RF)	**0.9936**	**0.9963**	0.9957	**0.9969**

Table 5.7: Comparison of our Novel Hybrid Approach with J. Knauth, XGBoost Classifier

	ACC	F1-S	PRE	REC
J. Knauth (AdaBoost)	0.9881	0.9896	0.9958	0.9835
Our Approach (XGBoost)	**0.9946**	**0.9969**	**0.9959**	**0.9978**

In [31], Rodríguez, J. et al. used different approaches by trying out datasets consisting of single types of users or different types of users and testing them on a specific type of user. To compare, we'll be taking the results from the experiments where they trained their models with different types of bots and took the mean of the results the model produced on single types of users. This is the closest approach that they have to us since we use different types of users for both training and testing while they test for only single types of users. Taking the mean of these results gives intuitively the same approach. Also, since they use AUC scores for their results, we'll use them. The 5 classifiers they also used in the testing that we've also used are AdaBoost, Random Forest (RF), Logistic Regression (LR), Naive-Bayes (NB), and k-NN. In 4 of the classifiers, AdaBoost, Random Forest, Naive-Bayes, and k-NN, our novel approach creates a huge difference in the AUC scores and performs much better than Rodríguez, J. et al. Only on the last classifier, Logistic Regression, the results are pretty similar although our model still does better than theirs. It should be noted that Logistic Regression is the best performing model of Rodríguez, J. et al., meaning only the second-best performing model of ours still does better than the best performing model of Rodríguez, J. et al. The comparison of our approach and Rodríguez, J. et al. can be seen in Table 5.8.

Table 5.8: Comparison of our Novel Hybrid Approach with Rodríguez, J. et al.

	AdaBoost	RF	LR	NB	k-NN
Rodríguez, J. et al.	0.812	0.804	0.903	0.712	0.745
Our Approach	**0.984**	**0.984**	**0.929**	**0.867**	**0.970**

5.3.3 Live Testing Results and Model Selection

In addition to comparing our results with other approaches in the literature, we still had the problem of selecting the best algorithm to use in our system. Looking back at the results we've achieved in Table 5.4, we can easily single out the best three classifiers XGBoost, AdaBoost, and Random Forest, which all have an accuracy score of 0.9934 or more with very well-rounded precision, recall, and other metric values. It's clear that one of these three is the best approach to use in our live system but there still is a decision on which one would be best suited for live predictions. To find the best model, we've tested the three models with the hand-picked dataset of 100 users we've created on the 8 evaluation metrics we've discussed in previous tests. The results from these tests can be seen in Table 5.9.

Table 5.9: Live Dataset Results

	ACC	ERR	PRE	REC	SPE	F1-S	FNR	FPR
XGBoost	0.75	0.25	1.0	0.5	1.0	0.6666	0.5	0.0
AdaBoost	0.78	0.22	0.9375	0.6	0.96	0.7317	0.4	0.04
Random Forest	0.89	0.11	0.9756	0.8	0.98	0.8791	0.2	0.02

XGBoost did well when finding humans as it correctly guessed all 50 of the human accounts but it did quite poorly on the bot as it only guessed 25 out of the 50 accounts correctly. AdaBoost did nearly the same on the humans as it correctly guessed 48 of the 50 accounts, although it did much better on bots as it guessed 30 out of the 50 bot accounts correctly. Lastly, Random Forest guessed 49 out of 50 human accounts correctly but made a huge improvement in the bot accounts as it guessed 40 of the 50 bot accounts correctly. Out of these tests, XGBoost, AdaBoost, and Random Forest had accuracies of 0.75, 0.78, and 0.89. After this, we also wanted to do some additional testing as the original dataset we picked contained 9 bots that were not malicious. By subtracting bots from the dataset and testing the three classifiers again, we had the results shown in Table 5.10.

Table 5.10: Live Dataset Results with Non-Malicious Bots Excluded

	ACC	ERR	PRE	REC	SPE	F1-S	FNR	FPR
XGBoost	0.7912	0.2087	1.0	0.5365	1.0	0.6984	0.4634	0.0
AdaBoost	0.8241	0.1758	0.9310	0.6585	0.96	0.7714	0.3414	0.04
Random Forest	0.9010	0.0989	0.9705	0.8048	0.98	0.88	0.1951	0.02

Two of the classifiers, XGBoost and AdaBoost, did better this time around as they found, respectively, 22 out of 41 and 27 out of 41 bots correctly. This can also be seen from both the accuracy and precision values going up for both. Random Forest, on the other hand, didn't have much change in terms of performance as it found 33 out of the 41 bots correctly. Also, its accuracy stayed similar.

Even though the results of XGBoost and AdaBoost went up with the exclusion of non-malicious bots, they still weren't close to the results Random Forest produced. Also, the accuracy of Random Forest stayed the same for both experiments, showing that it did better all around. Because of these reasons, we chose the random forest classifier to be used in our live system.

5.3.4 Discussion About the Interim Report

The main goal we had discussed in our interim reports was the goal to be able to build a whole system. We also had some subset of goals under the main goal. In the end, we were able to build a whole system that enabled users to predict a Twitter account and display the results on a website. Some of the main goals we had divided under our main goal were an easy-to-use interface, implementation of a machine learning algorithm, publication of our system, etc. Building the system from scratch, we were able to meet these goals one by one to create our whole system. In the end, the only goal we weren't able to meet was building a browser extension, which was a removed feature due to server, money, and coding constraints that the way browser extensions were developed put on us. Other than this, we were able to meet all the goals we had discussed in our interim report. A complete list of goals given in our interim report and the ones we've met are given on the next page. The goals we've achieved are given in bold.

1. **Implementation of a novel machine-learning algorithm to detect Twitter bots.**
2. Implementation of a browser extension.
3. **Implementation of a website.**
4. **Implementation of a unique and easy-to-use user interface.**
5. **Publication of our system.**
6. **Our final and main goal is to be able to implement all of the things listed above and create a whole system.**

When it comes to the evaluation criteria, we were able to meet all of them except for one criterion about the publication of a browser extension. Since this part of the system was never done, we weren't able to meet that goal. We were able to achieve accuracy, precision, and recall values above %92, had unit tests that covered %40 the code, used a dataset that consisted of at least 5000 users, and created an easy interface that was adaptable by a user in less than 2 minutes, etc. The full list of evaluation criteria we had discussed in the interim report is also given below as a list. The only criteria we weren't able to achieve was 11, which was the publication of the browser extension. The criteria we've achieved are shown in bold.

1. **The system should be usable more than %85 of the time.**
2. **The system should not have a downtime of more than 24 hours in the case of a failure.**
3. **The system should be scalable from 1 to 10 to 100 users.**
4. **The bot detection algorithm should provide no less than 0.92 accuracy, precision, and recall.**
5. **The bot detection algorithm should display results in no more than 30 seconds.**
6. **Unit tests on the code should pass at least for %40 of code coverage.**
7. **Machine learning algorithms should be trained with data sets containing at least 5000 users.**
8. **The final system should at least have an MTTF of 100 hours.**
9. **System throughput should not be less than 1000 operations per second.**
10. **The system should be able to handle at least 10 users simultaneously.**
11. The system should be deployed on at least 1 modern browser extension.
12. **User adaptation to the system should not take more than 2 minutes.**

6 Conclusion and Future Work

Social media have become the main source for people for different activities since the rise of the internet. The big data generated daily in such platforms inevitably results in malicious content that is generated by bot accounts. In this project, we were able to create a system that can detect such bots on Twitter. Firstly, we introduced an all-around review of the machine learning models discussed in the literature by categorizing them into categories, unsupervised, semi-supervised, and supervised algorithms. We also presented a detailed review of the types of bots that can be encountered in a social setting. We then discussed our approach to solve the problem of detecting bots on Twitter, which would be a system that predicted Twitter accounts based on supervised machine learning. Our approach was explained using the data, structural and dynamic models. The data model of our system explained the dataset that would be used in the project while also explaining the final system we wanted to achieve. In the structural model, we discussed the approaches used to achieve the supervised machine learning model that would be used in the system. We sought out approaches, [18] and [31], to create similar versions of them to start. Then, adding our intuition to the inspirations we got from [18], [27], and [31], we created a novel hybrid approach to create a machine learning model. As for the dynamic model, we further explained our system by providing our main use case and its explanation, with the help of a sequence diagram. We also added the interface of our system with its explanations. The experimentation environment and the methods, such as creating a basis for experimentation, and live testing, used were then discussed. Finally, the comparative evaluation and discussion of the experiment's results were given to compare our research with the existing literature, decide on the final model to use in our system, and discuss our goals and evaluation criteria given in the interim report. With our system, we were able to achieve the contributions which were feature extraction on a new set of features, higher results of accuracy, precision, and recall compared to past studies, a unique user interface that is easy to understand and use, and explanations of our detection methods and their decisions for product integrity.

As feature work, our approach can be extended in many different areas. One of these areas is the dataset we've used. The Cresci-17 dataset only consisted of three types of bots which were fake followers, traditional bots, and social spambots. Including different types of bots in this dataset like political bots, pron bots, botnets, etc., could be used as a basis for further experimentation and a more generalizable model. Also, on the topic of datasets, the live dataset we used can be extended further to have more bots than it does in the research. Since all of the Twitter accounts were randomly hand-picked by us, we only resorted to a dataset consisting of 50 bots and 50 humans. This dataset can be made bigger to increase the number of bots and humans, and also, to increase the bots types as well.

Another featured work would be on further extending our approach to creating a machine learning model. With the new Twitter API v2, Twitter gives access to new features it has, such as spaces. These newly accessible features that aren't used in any past research, might be found useful in detecting state-of-the-art bot models which use different algorithms than the more generally expected bots such as fake followers, spambots, etc. Our approach can also be extended to other areas. One of these areas is the classification results we use. In our model, the predictions are given as binary classification results. These results can be made to have other criteria such as confidence intervals, where the result would be given with a confidence value indicating how likely it's for that result to be accurate.

7 References

[1] M. Mohsin., “10 social media statistics you need to know in 2021,” 2020. [Online]. Available: https://www.oberlo.com/blog/social-media-marketing-statistics.
[2] G. Johar, Y. Jun, and R. Meng, “Perceived Social Presence Reduces Fact Checking,” *Proceedings of the National Academy of Sciences*, vol. 114, May 2017, doi: 10.1073/pnas.1700175114.
[3] D. Sayce, “Number of tweets per day 2019,” 2019. [Online]. Available: https://www.dsayce.com/social-media/tweets-day/#:~:text=Every%20second%2C%20on%20average%2C%20around%206%2C000%20tweets%20are%20tweeted%20on,200%20billion%20tweets%20per%20year.
[4] S. Wojcik, S. Messing, A. Smith, L. Rainie, and P. Hitlin, “Bots in the Twittersphere,” 2018.
[5] C. Shao, G. Ciampaglia, O. Varol, A. Flammini, F. Menczer, and K.-C. Yang, “The spread of low-credibility content by social bots,” *Nature Communications*, vol. 9, Nov. 2018, doi: 10.1038/s41467-018-06930-7.
[6] news18, “Facebook removed 22.5 million hate speech content, 1.5 billion fake accounts in q2 2020,” 12 2020. [Online]. Available: https://www.news18.com/news/tech/facebook-removed-22-5-million-hate-speech-content-1-5-billion-fake-accounts-in-q2-2020-2779339.html
[7] M. IQBAL, “Twitter revenue and usage statistics (2022),” 2022. [Online]. Available: https://www.businessofapps.com/data/twitter-statistics/
[8] O. Varol, E. Ferrara, C. A. Davis, F. Menczer, and A. Flammini, “Online Human-Bot Interactions: Detection, Estimation, and Characterization,” *ArXiv*, vol. abs/1703.03107, 2017.
[9] Gilani, Zafar, Reza Farahbakhsh, Gareth Tyson, Liang Wang, and Jon Crowcroft. "Of bots and humans (on twitter)." In Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017, pp. 349-354. ACM, 2017.
[10] S. Haustein, T. Bowman, K. Holmberg, A. Tsou, C. Sugimoto, and V. Larivière, “Tweets as impact indicators: Examining the implications of automated ‘bot’ accounts on Twitter: Tweets as Impact Indicators: Examining the Implications of Automated ‘bot’ Accounts on Twitter,” *Journal of the Association for Information Science and Technology*, vol. 67, Oct. 2014, doi: 10.1002/asi.23456.
[11] S. Cresci, R. Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, “Fame for sale: Efficient detection of fake Twitter followers,” *Decision Support Systems*, vol. 80, Sep. 2015, doi: 10.1016/j.dss.2015.09.003.
[12] A. Dhiman and D. Toshniwal, “An Unsupervised Misinformation Detection Framework to Analyze the Users using COVID-19 Twitter Data,” in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 679–688. doi: 10.1109/BigData50022.2020.9378250.
[13] Y. Fitriani, S. Sumpeno, and M. H. Purnomo, “Classifying Twitter Spammer based on User’s Behavior using Decision Tree,” in *2019 Asia Pacific Conference on Research in Industrial and Systems Engineering (APCoRISE)*, 2019, pp. 1–7. doi: 10.1109/APCoRISE46197.2019.9318872
[14] M. Debashi and P. Vickers, “Sonification of Network Traffic for Detecting and Learning About Botnet Behavior,” *IEEE Access*, vol. 6, pp. 33826–33839, 2018, doi: 10.1109/ACCESS.2018.2847349.

[15] D. Assenmacher, C. Grimme, and L. Clever, *Changing Perspectives: Is it Sufficient to Detect Social Bots?* 2018.
[16] G. Wang *et al.*, “Social Turing Tests: Crowdsourcing Sybil Detection,” May 2012.
[17] A. Derhab, R. Alawwad, K. Dehwah, N. Tariq, F. A. Khan, and J. Al-Muhtadi, “Tweet-Based Bot Detection Using Big Data Analytics,” *IEEE Access*, vol. 9, pp. 65988–66005, 2021, doi: 10.1109/ACCESS.2021.3074953.
[18] J. Knauth, “Language-Agnostic Twitter-Bot Detection,” in *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, Sep. 2019, pp. 550–558. doi: 10.26615/978-954-452-056-4_065.
[19] O. Loyola-González, R. Monroy, J. Rodríguez, A. López-Cuevas, and J. I. Mata-Sánchez, “Contrast Pattern-Based Classification for Bot Detection on Twitter,” *IEEE Access*, vol. 7, pp. 45800–45817, 2019, doi: 10.1109/ACCESS.2019.2904220.
[20] S. PV and M. Saira Bhanu, “UbCadet: detection of compromised accounts in twitter based on user behavioural profiling,” *Multimedia Tools and Applications*, vol. 79, Jul. 2020, doi: 10.1007/s11042-020-08721-z.
[21] Z. Alom, B. Carminati, and E. Ferrari, “Detecting Spam Accounts on Twitter,” in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 2018, pp. 1191–1198. doi: 10.1109/ASONAM.2018.8508495.
[22] F. N. Pakaya, M. O. Ibrohim, and I. Budi, “Malicious Account Detection on Twitter Based on Tweet Account Features using Machine Learning,” in *2019 Fourth International Conference on Informatics and Computing (ICIC)*, 2019, pp. 1–5. doi: 10.1109/ICIC47613.2019.8985840.
[23] S. devi and K. Subbaraj, “#CycloneGaja-Rank based Credibility Analysis System in social media during the crisis,” *Procedia Computer Science*, vol. 165, pp. 684–690, Jan. 2019, doi: 10.1016/j.procs.2020.01.064.
[24] A. Dorri, M. Abadi, and M. Dadfarnia, “SocialBotHunter: Botnet Detection in Twitter-Like Social Networking Services Using Semi-Supervised Collective Classification,” in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 496–503. doi: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00097.
[25] A. T. Kabakus and R. Kara, “‘TwitterSpamDetector’: A Spam Detection Framework for Twitter,” *Int. J. Knowl. Syst. Sci.*, vol. 10, pp. 1–14, 2019.
[26] M. Kouvela, I. Dimitriadis, and A. Vakali, “Bot-Detective: An explainable Twitter bot detection service with crowdsourcing functionalities,” Nov. 2020, pp. 55–63. doi: 10.1145/3415958.3433075.
[27] M. Fazil and M. Abulaish, “A Hybrid Approach for Detecting Automated Spammers in Twitter,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2707–2719, 2018, doi: 10.1109/TIFS.2018.2825958.
[28] S., A., Han, T., Wu, W., Song, H., & Kumar, A. (2020). Twitter spam account detection based on clustering and classification methods. *The Journal of Supercomputing*, *76*. https://doi.org/10.1007/s11227-018-2641-x
[29] Shevtsov, A., Oikonomidou, M., Antonakaki, D., Pratikakis, P., Kanteraks, A., Fragopoulou, P., & Ioannidis, S. (2022). Discovery and Classification of Twitter Bots. *SN Computer Science*, *(2022) 3:255*. https://doi.org/10.1007/s42979-022-01154-5
[30] Varol, O., Ferrara, E., Davis, C. A., Menczer, F., & Flammini, A. (2017). Online Human-Bot Interactions: Detection, Estimation, and Characterization. *ArXiv*, *abs/1703.03107*.

[31] Rodríguez, J., Mata Sánchez, J., Monroy, R., Loyola-González, O., & López-Cuevas, A. (2020). A one-class classification approach for bot detection on Twitter. *Computers & Security*, *91*, 101715. https://doi.org/10.1016/j.cose.2020.101715
[32] K. Yang, "Bot Repository.", https://botometer.osome.iu.edu/bot-repository/datasets.html (accessed: Jun. 14, 2022)
[33] Yang, K.-C., Ferrara, E., & Menczer, F. (2022). *Botometer 101: Social bot practicum for computational social scientists*. arXiv. https://doi.org/10.48550/ARXIV.2201.01608