Kieran Brahney

Social Bot Identification on Twitter

B.Sc. Computer Science with Industrial Experience

21/03/2013

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the Department's use of plagiarism detection systems in order to check the integrity of assessed work.
I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date: _______________

Signed: _______________

ABSTRACT.

In this report, Twitter, a popular micro-blogging site, is studied to identify new and existing features that can be used to identify bot accounts. A real labelled dataset is created, with a demonstrated inter-rater agreement, alongside a larger unclassified dataset. The evaluation applies four different classifiers, and uses a combination of features indicated by prior research and newly identified semantic features to create a detection system of optimised features, which is shown to be accurate in identifying bot accounts.

TABLE OF CONTENTS

WORKING DOCUMENTS

A summary of the working documents for this report can be found at:

http://www.lancaster.ac.uk/ug/brahney/FYP/

Further links will be available from this page to each individual working document.

1 INTRODUCTION

Online social media hit the world by storm in the early 2000's with sites such as Friendster and MySpace, ever since then usage of social media has been on the increase and only recently has it reached a truly global audience. Twitter, the network this report focuses on, was founded by Jack Dorsey a thirty-seven year old Computer Programmer in June 2006 as a microblogging platform with the intentions of providing up-to-the minute information about global trending events. The difference between Twitter and other social networks is that users interact with each other using short text based posts limited to 140 characters which are known as tweets. A tweet can include hashtags, these are a short text based string prefixed by a # character used to associate the tweet to a given topic. When many users interact with the same topic it can lead to what is known as a Trending Topic, such that it is considered popular. Trending Topics are associated with the geographic location of the user posting and enable one to search for all the users posting to a given topic. Relationships on Twitter differ from prior and other current social networks in that it consists of followers. These are people who subscribe to receive all the tweets from a given user(s), the relationship can become bidirectional when two users follow each other, they are then known as *friends*.

As of 1st Jan 2014 there are approximately 645 million Twitter users (Statistic Brain, 2014), the growing popularity has meant it has become an ideal home for automated accounts or bots. Automation is a double edged sword; on the one hand legitimate bots provide instant information such as that from news feeds. On the other hand, malicious bots take advantage of Twitter's social model by spreading spam (malware/advertising) through Trending Topics or overwhelming users with notifications. Twitter itself does not distinctly check for automation, only requiring the recognition of a CAPTCHA during registration. This begs the question, despite the rise and reliance on Twitter data for providing up-to-the-minute information can one truly trust the content and distinguish between automated and legitimate messages. As such there have been a number of a research efforts into determining the difference between bots and humans on the social network. However, these approaches have yet to look into the effects of language analysis techniques and also no existing work provides up-to-date, reliable, data sets of bot users.

The aim of this project is to design and implement a classification system that can accurately and efficiently differentiate between bots and humans on the social network, Twitter. This root aim can be further divided into a number of sub-goals:

- Discover state-of-the-art approaches for detecting bots by reviewing existing research and understand how to build upon their limitations
- Design and implement a data gathering system to build a Twitter data set that can be annotated via web interface as to whether a user is a bot or not
- Identify new features or techniques to extend upon the existing research base
- Apply machine learning algorithms to the data-set using a combination of previously used (state-of-the-art) and new features
- Analyse the results of various algorithms and feature sets using statistical tests

In the report, before detailing the proposed system, existing research methods are scrutinized to discover current state-of-the-art approaches, and their limitations, for the detection of bots on Twitter or other areas of computing, together with the methods they employed with regard to data collection.

The report is broken into two key sections, data gathering and account classification. The data gathering section of this report focuses on the key design decisions in building a gold standard data set due to lack of existing work providing up-to-date data sets of bot users. Furthermore, the design and implementation of an annotation system to enable the manual classification of the data set that could be used to evaluate the approach against. Contributions to the community include a pre-classified data set with proven high inter-rater agreement, also a much larger unclassified data set. The account classification section will then focus on the feature sets employed for use with various Machine Learning algorithms. New feature contributions will be analysed against existing state-of-the-art using statistical tests to see how these affect the overall classification result. To conclude, a description of the overall results will be presented along with proposals for further work in this area.

2 BACKGROUND

Detection of automation has been widely studied for a number of decades across the computing industry, looking at automations such as gaming bots and web-based spam bots. Previous work into spam detection has looked into email and web but until 2010 not much of this research had been applied to social networks such as Twitter. This section will summarise the existing research of state-of-the-art approaches to detecting bots on Twitter and then later, analyse their limitations (Section 2.2) with the aim of improving on them in this report.

2.1 Relevant Work

2.1.1 Detecting Spam Bots in Online Social Networking Sites: A Machine Learning Approach

(Wang, 2010) proposed a machine learning approach to distinguishing spam bots in online social networking sites. The research performed data collection using the public_timeline API, selecting 20 non-protected users whom had set a custom user icon. For each of the 20 returned users their account details, 20 most recent tweets and a list of their followers/friends IDs were extracted via separate API calls. Out of this data set (Wang, 2010) manually labelled 500 accounts into spam or not spam by reading the users 20 most recent tweets and checking the number of followers/friends.

The research method used 3 graph-based features (number of friends, number of followers, follower/friend ratio) and 3 content-based features (duplicate tweets, total URLs, total replies/mentions) for the machine learning algorithms. These features were applied to four classifiers: Decision Tree, Neural Networks, Support Vector Machines (SVM) and Naive Bayes, using 10-fold cross validation. The results of the experiment were proven against the F-Measure produced for each classifier, with Naive Bayes performing best at 91.7% accuracy.

2.1.2 Detecting Spammers on Twitter

(Benevenuto, et al., 2010) took a similar approach to (Wang, 2010) in detecting bots on Twitter but on a much larger scale. The data gathering process involved crawling all Twitter user IDs from 0 to 80 million, returning 54 million active accounts. 8207 of these accounts were selected for pre-classification based on whether they had posted tweets in 3, largely discussed, trending topics from 2009. To further the dataset, users were randomly selected, to include those of whom posted at least one tweet containing a URL together with one of the 3 trending topic terms. The selected data was then made available via a website for volunteers to classify the accounts. Each account was classified by 3 volunteers to reduce the possibility of human error.

Two feature sets (each consisting of 10 features) were determined for use on machine learning algorithms, these being content based and user behaviour. The most promising features for detecting spammers were found to be high use of spam words, URLs, hashtags, with a high follower/friend ratio and typically a young account. The 20 features were then applied to the libSVM classifier using 5-fold cross validation. Experimentation results revealed 70% accuracy in determining spammers with 96% non-spammers, though there was a clear trade-off between incorrectly classifying non-spammers which could be manipulated through the cost mechanism of SVM.

2.1.3 Detecting Automation of Twitter Accounts: Are you a Human, Bot or Cyborg?

(Chu, et al., 2012) aimed to characterise the differences between Humans, Bots and Cyborgs. The data gathering process made use of a depth-first search approach. Firstly 5 users were randomly selected as seeds for the system, for each user the system would look at the follower list and traverse to a depth of 3. Similar to the work of (Wang, 2010), the public_timeline API was also used to get the 20 most recent global tweets to help diversify the user pool. In total 512,000 users were collected using both methods simultaneously. Using the principle behind the Turing test 6,000 users were manually classified into the 3 distinct categories based on intelligence, originality of tweets and types of URL present (malicious).

The pre-classified data set was applied to the Random Forest Machine Learning algorithm with features including: entropy of tweet intervals, source of tweets, URL blacklist checking, and account reputation. The computed features were plugged into the Weka explorer using 10 fold cross validation, yielding results on average of 96% accuracy. It was concluded that bots and cyborgs exhibit regular patterns with low entropy and the most accurate features were tweet source and URL blacklist checks.

2.1.4 Humans and Bots in Internet Chat: Measurement, Analysis, and Automated Classification

Internet chat is a popular concept that has been around for a number of years, being employed in games and commercial chat networks. (Gianvecchio, et al., 2011) studied the characteristics of automated accounts in Yahoo! chat. In August and November 2007 1440 hours of chat logs were randomly gathered from popular chat rooms. Similar to (Chu, et al., 2012) a variation of the Turing test was employed for manual classification of chat users into human, bot and ambiguous.

Statistical and conversational analysis revealed that bots often send messages at regular intervals, replay other user’s messages and use synonyms to avoid blacklisted phrases. Firstly, an entropy based classifier was used on the message size and inter-message delay to build a human or bot corpus. Secondly, a Bayesian classifier (CRM114 Discriminator) was used against the content of chat messages. It was concluded the entropy solution was able to detect unknown bots but it took a large amount of data to correctly classify it, whereas the Bayesian classifier with the help of the bot corpus provided by the entropy classifier was quick and accurate.

2.1.5 Detecting Social Spam Campaigns on Twitter

In 2012 (Chu, et al., 2012) used multiple features and Machine Learning to identify spam campaigns on Twitter. From February to April in 2011 more than 50 million tweets and around 22 million accounts were gathered to form a data set. This was achieved using a combination of the Twitter streaming and search APIs, respectively. The study specifically focused on content containing URLs as an indicator for possible spam activity, converted any short URLs to its final destination. A clustering algorithm was then used to cluster tweets that shared the same URL into spam campaigns. Any unknown campaigns were finally labelled by a human leaving a total pre-labelled data set of 1324 campaigns.

The classification system implemented the classifier based on the Random Forest algorithm. A variety of features were used; the most accurate identified as: account diversity ratio, timing entropy, URL blacklist checks and follower/friend ratio. Content Semantic Similarity was a new feature implemented by Chu et al. it sought to find duplicate or similar content across a user’s tweets. The results of the classification are compared against 7 other classifiers with Random Forest performing best at 94.5% accuracy.

2.1.6 Detecting Spammers on Social Networks

(Stringhini, et al., 2010) analysed the extent to which spammers infiltrated 3 popular social networks, Facebook, MySpace and Twitter. A large data set was gathered by creating 300 “honey-profiles” on each of the networks. These accounts would then log any activity with the account, including accepting friend requests, and any information sent to the account for a period of 12 months from June 2009 to June 2010. In the case of Twitter, the data set was then manually labelled by the researchers into those that were legitimate and those that were spam.

The classification system made use of the Weka Machine Learning framework with Random Forest employed as the classifier. A general feature set was implemented for all networks, this included: follower/friend ratio, use of URLs, message similarity, number of tweets and number of friends. The approach concluded that out of 15,932 accounts detected as spammers and reported to Twitter, only 75 were reported back as being false positives.

2.1.7 Spam Detection on Twitter Using Traditional Classifiers

(McCord & Chuah, 2011) measured spam detection on Twitter using user and content-based features. A data set was formed by crawling the Twitter API gathering account information together with 100 of their most recent tweets. 1,000 users were then randomly selected and labelled into two classes, ‘spam’ and ‘non-spam’, by the researchers to use as a ground truth.

The ground truth was run against 4 traditional classifiers: Random Forest, Support Vector Machine, Naive Bayes and k-Nearest Neighbours. The features implemented for use in these classifiers were broken down into user and content-based features. User-based features included follower/friend ratio, and the distribution of tweets over a 24 hour period. The content-based feature included the number of URLs, interactions, re-tweets, hashtags, the use of spam words and also the length of the tweet. The result of these features yielded 95.7% accuracy when the Random Forest classifier was used.

2.1.8 Scaling-Laws of Human Broadcast Communication Enable Distinction between Human, Corporate and Robot Twitter Users

(Tavares & Faisal, 2013) looked at statistical laws dictating the timing of human actions in communication decisions. The researchers built their data set by crawling the Twitter API, gathering account data together with a total of 800 tweets per account. The crawler was provided with a list of accounts to process which had been manually selected by the researchers. In total, the study collected over 160,000 thousand tweets; a subset of which were manually classified into 3 user categories: Personal, Managed, and Bot-controlled based on time intervals between posts.

To test the hypothesis of differentiating based on timing intervals, two probabilistic inference algorithms were implemented. Firstly, a Naive Bayesian classifier looked at features such as the time of day of each tweet, the intervals between tweets, and the number of tweets per day. This classifier yielded between 75 - 84% accuracy. Secondly, a prediction algorithm looked to estimate the time of a user’s next tweet. The results showed that it was possible to reliably distinguish between the three categories based on timing intervals alone.

2.2 Limitations

(Chu, et al., 2012; Stringhini, et al., 2010) presented the idea of a feature in the form of content semantic similarity. This was the only of the related works to attempt analysis of the language in the tweet, by determining those of similar context as opposed to the simpler Levenshtein distance approach presented by (Wang, 2010). In this report, section 4 will detail new features improving upon the lack of language analysis in the form of sentiment and comparing tweet against hashtag terms.

A number of the related works described above did not employ an inter-rater agreement when pre-classifying their data set, instead they manually classified the accounts themselves. The lack of a diverse volunteer set for the classification of accounts may have led to the introduction of bias in the process of determining whether accounts were bot or a human, and suggest the results may be unreliable for assessment. In this report, section 3 will detail the use of 3 volunteers to label each account, together with the inter-rater agreement formed on the basis of their opinions.

(Benevenuto, et al., 2010) did however use an inter-rater agreement and also built a website to allow random members of the public to classify the data set. This being said, in Benevenuto et al.'s approach, and all other related works, the agreement between volunteers was not proven via statistical tests, thus the reliability is somewhat questionable. Section 3 of this report will describe how the Cohen's kappa statistic will be employed in order to determine the strength of agreement between volunteers and hence ensure a degree of reliability.

The number of classifiers used varied between the related works. However (Benevenuto, et al., 2010; Chu, et al., 2012; Gianvecchio, et al., 2011; Stringhini, et al., 2010; Tavares & Faisal, 2013) specifically focus on one classifier. Whilst there is a reason behind this approach, it could be considered a limitation in that it may generate a better solution using additional classifiers. Section 4 will then detail how results vary across 4 traditional classifiers utilising different feature sets.

Some of the data gathering approaches such as that of (Wang, 2010) only gathered a limited data set. For example, (Wang, 2010) only gathered a total of 20 tweets per user. The average Twitter user has more than 20 tweets[1] and thus such a small sample of tweets cannot be assumed to provide an accurate representation of an account. Section 3 of this report, will describe how the maximum possible number of tweets will be gathered from the API.

[1] Smith, C., 2014. *116 Amazing Twitter Statistics (Updated February 2014).* [Online]
Available at: http://expandedramblings.com/index.php/march-2013-by-the-numbers-a-few-amazing-twitter-stats
[Accessed 16/3/2014]

3 DATA GATHERING AND ANNOTATION

Data gathering is the process of crawling Twitter to produce a data set against which the classification models can be run on. Prior to undergoing the task of producing a system which can gather, manage and enable classification of the accounts a search for existing data sets took place. This research involved contacting (Wang, 2010; Benevenuto, et al., 2010; Chu, et al., 2012) to enquire if their data set was publicly available. In the case of (Benevenuto, et al., 2010) a data set was publicly available however due to a request from Twitter information about users and tweets had to be removed. The approach outlined in section 4 of this report very much relies upon this data to build the feature set. Thus unfortunately, it soon became apparent that there were very few publicly available data sets that had already been classified as Human/Bot. This difficulty led to the decision to design and implement a system that could gather the required data, with a separate system being employed for the classification of the acquired data. This approach ensured a bespoke, gold-standard, data set could be built with proven inter-rater reliability thus improving upon the limitations of previous methods identified in section 2.2.

The data gathering process was divided into two separate functions, firstly a standalone program to crawl the Twitter API and scrape data; followed by another more comprehensive system for the annotation of the accounts. All parts of the system were developed using a combination of PHP and MySQL for processing and web technologies (HTML, CSS, JavaScript) for any front-end interfaces. Whilst these may not be the most efficient in terms of processing or data storage in the modern climate, they were readily available with large support communities and did not have any problems with the amount of data that this project was processing.

3.1 Twitter Crawler

3.1.1 Design

The crawler was composed of two standalone PHP programs, running on a virtual private server (VPS.) The first program, known from here on as *the Queue*, crawled the Twitter API for account data on a periodic basis and stored this in a MySQL database. The second program, from here on known as *the Poller*, polled data from the queue, making further calls to the API to gather account details and then also stored this in the MySQL database. Figure 3-1 shows the architecture of the crawler and the various components involved, database views are described in their appropriate sections.

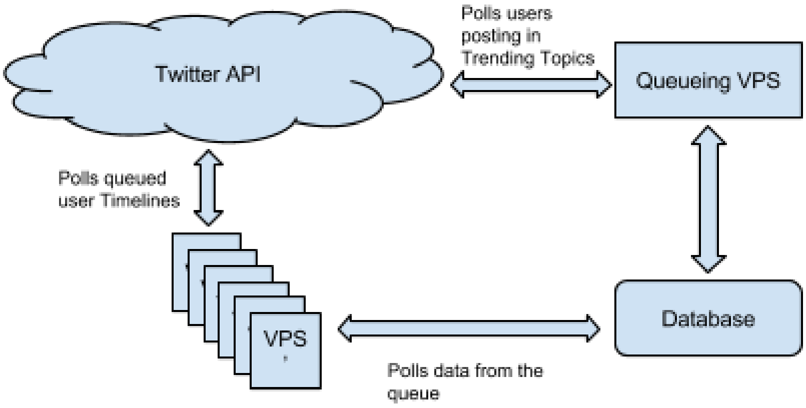

Twitter API
Polls users posting in Trending Topics
Queueing VPS
Polls queued user Timelines
VPS
Database
Polls data from the queue

Figure 3-1 Twitter Crawler System Architecture

A key design decision regarding accessing the Twitter API was understanding their rate limiting system. Rate limiting is the process of restricting the number of requests any user can make to the API per rate limiting window. As of version 1.1 of the API, Twitter adjusted the rate limiting window to be divided into 15 minute intervals. Typically the amount of requests varies depending on the particular API function that one wishes to access, these will be detailed later on.

The queue

The queue data store was emulated using a single MySQL table consisting solely of Twitter account IDs. The objective of the queuing program was to gather a diversified set of account data across a range of users. The following shows a breakdown of how the program worked (see Table 3-1):

1. Lookup the top UK Trending topics. The decision to use the UK, when other locations are available, was due to tweets containing different languages which would ultimately interfere with any attempts at using Natural Language Processing during classification.
2. Search for tweets existing within the returned trending topics, again, limited to English only.
3. Lookup the returned users' followers.

Step	API Function	Rate limit	Number of calls made	Description of returned data
1	GET trends/place[2]	15/user 15/app	1	10 trending topics
2	GET search/tweets[3]	180/user 450/app	10	100 tweets matching search query
3	GET followers/ids[4]	15/user 15/app	15	Up to 5,000 user IDs

Table 3-1 The queue API Functions

Table 3-1 shows that for each of the ten trending topics, 100 associated tweets were returned resulting in potentially 1,000 tweets and associated user data. One key limitation of this design was that out of this 1,000 it was only possible to lookup 15 of those users' followers (due to rate limiting in step 3). This means that the maximum amount of accounts that can be returned by the program, in one go, is $(10 \times 100) + (15 \times 5{,}000) = 76{,}000$ accounts. The decision to add the follower ID lookup was to speed up the crawling process and thus no real data is lost by not having the ability to further lookup the followers of the remaining $1000 - 15 = 985$ accounts.

[2] Twitter, 2013. GET trends/place | Twitter Developers. [Online]
Available at: https://dev.twitter.com/docs/api/1.1/get/trends/place [Accessed 2/3/2014].

[3] Twitter, 2013. GET search/tweets | Twitter Developers. [Online]
Available at: https://dev.twitter.com/docs/api/1.1/get/search/tweets [Accessed 2/3/2014].

[4] Twitter, 2013. GET followers/ids | Twitter Developers. [Online]
Available at: https://dev.twitter.com/docs/api/1.1/get/followers/ids [Accessed 2/3/2014].

The Poller

The objective of the Poller was to function once the queue had stored sufficient data. It polled data from the queue in groups of 300 and made a lookup for the particular user's timeline, the returned data was then stored in a User, and Tweets table. Table 3-2 justifies polling in groups of 300 because application authentication meant that much more data could be processed per rate limit window.

Step	API Function	Rate limit	Number of calls made	Description of returned data
1	GET statuses/user_timeline[5]	180/user 300/app	300	Up to 3,200 tweets and various user data

Table 3-2 The Poller API Functions

3.1.2 Implementation

Authentication with the Twitter API requires OAuth, an open protocol to allow secure authorization in a simple and standard way. A number of OAuth libraries exist for PHP[6] however one that I am particularly familiar with is tmhOAuth[7]. The library was used to deal with initial authentication requests to the API. To further the use of the library a number of wrapper functions were created in order to keep the code clean but also make it easy to gather the required data and parse it in a clean manner.

The User table consisted of the basic account properties, this included: Twitter account ID, account name, description, creation date, location, total followers, total friends and whether the default profile picture was set. The Tweets table consisted of all of a user's tweets, this included: tweet ID, account ID, tweet contents, date/time, source, whether or not the tweet was retweeted and a retweet count. In both instances the date/time was converted to UTC and stored as the DATETIME data type. A relationship links the two tables by account ID.

[5] Twitter, 2013. GET statuses/user_timeline | Twitter Developers. [Online] Available at: https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline [Accessed 2/3/2014].
6 Twitter, 2014. Twitter Libraries | Twitter Developers. [Online] Available at: https://dev.twitter.com/docs/twitter-libraries#php [Accessed 10/3/2014].
[7] GitHub, 2013. themattharris/tmhOAuth. [Online] Available at: https://github.com/themattharris/tmhOAuth [Accessed 10/3/2014].

3.1.3 The System in Operation

Before the Twitter crawler could operate, an API account had to be created by registering a Twitter account and signing into the development portal[8]. In the design section it was noted that Twitter applications get higher rate limit windows than single user access, thus in the development portal an independent application was registered for the API account. The application provided authentication keys for the Twitter API, these were plugged into the Twitter crawler code before operation.

The standalone queue program described in the design section was placed onto a VPS and executed once every 15 minutes using the Linux cron[9], for example:

```
*/15 * * * * php /home/kieran/Twitter/QueueUsers.php
```

Once the queue had sufficient account data (100,000+) it was stopped and the *Poller* was set to execute once every 15 minutes. In order to speed up the *Poller*, this process was then repeated across 5 independent VPS'. As mentioned in Table 3-2 this allowed for $5 \times 300 = 1{,}500$ accounts to be processed from the queue every 15 minutes.

The standalone polling program described in the design section ran in parallel to the queue but in this instance only ran from the database server once every 15 minutes. The data gathering process ran for two weeks from the 13/11/2013 gathering a total of 75 million tweets and 485,000 users.

[8] Twitter, 2014. Twitter Developers. [Online]
Available at: https://dev.twitter.com [Accessed 10/3/2014].

[9] ArchLinux, 2014. *cron - ArchWiki*. [Online]
Available at: https://wiki.archlinux.org/index.php/cron [Accessed 10/3/2014]

3.2 Annotation System

3.2.1 Design

The annotation system was a key part of the data gathering process. The main aim of the annotation system was to provide an easy-to-use web interface for the classification of previously gathered account data. Ease-of-use of the system would be key to enabling the volunteer to quickly make informed decisions about a given account thus allowing efficient use of a volunteer's time.

The system was broken down into two components, authentication and annotation. Upon first visit to the system a volunteer was required to register an account, providing a unique email address and password. Registration of an account was a required process so that annotation actions can be tied back to a particular volunteer. It was thus important that the accounts were secured by password authentication so it is ensured that actions are the volunteers own.

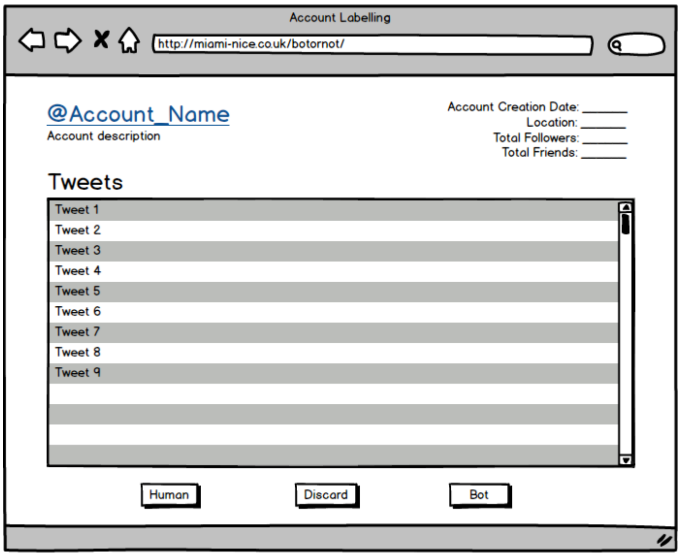


Figure 3-2 Annotation Interface Mockup

Once authenticated the volunteer was immediately presented with an account to annotate as illustrated in Figure 3-2. The system enabled a volunteer to classify an account as either bot, human or discard. A discard option was provided as Twitter makes it apparent in the API documentation[10] that language detection is best effort, thus there was the possibility that some tweets may not be in English which will then, later interfere with Natural Language Processing techniques.

The annotation system required that a total of 3 volunteers label each account. The purpose behind this approach is to form a consensus based on the 3 provided verdicts, for example: user A states bot, user B states bot and user C states human; the consensus here would be bot. This is a major enhancement upon previous research efforts such as that of (Wang, 2010) who labelled the dataset personally. The present approach overcomes that by allowing a diversified set of volunteers to annotate as many accounts as they please. Moreover, it should also provide greater agreement regarding annotations based on more than one user's opinion.

3.2.2 Implementation

The annotation system was built using a combination of PHP for the backend activity and HTML/CSS/JavaScript for the user interface. The user interface made use of an open source web development framework called Bootstrap[11]. The components provided by Bootstrap are used to quickly and easily create well-designed interfaces without the hassle of composing the CSS/JavaScript oneself. For example, the annotation interface was specifically engineered using Bootstrap to fit on the vast majority of screen resolutions without the need

[10] Twitter, 2013. GET search/tweets | Twitter Developers. [Online]
Available at: https://dev.twitter.com/docs/api/1.1/get/search/tweets [Accessed 2/3/2014].

[11] *Bootstrap.* [Online]
Available at: http://getbootstrap.com/ [Accessed 13/3/2014]

for scrolling. This key design decision ensured that all information was available to the user on one screen and the user could quickly make an informed decision as to how to classify a given Twitter account.

In order for the annotation system to function a number of SQL tables were required:-

1. Login data:
   As described in the design section the login data required an e-mail address, password and optional last activity timestamp. The password was stored encrypted using the MD5 hashing algorithm with a maximum length of 32 characters. Encrypting the password data was a requirement of the system due to the public facing nature of the annotation system.

2. Accounts to annotate
   This table consisted of the Twitter account ID, who had annotated the account and the total number of users who had annotated the account. Who had annotated the account was a character field storing a serialized array of the user ID from the login data table. The total number of times the account had been annotated was also provided to avoid calculating this every time.

3. Annotation results
   The annotation results consisted of the Twitter account ID, the user id (from the login table) and the result, i.e.: what they classified the account as human, bot, or discard. This classification field could have been simplified into an integer to reduce the amount of space taken by the field but this table only consisted of 3,000 rows (3 x 1,000 accounts to label.)

The accounts to annotate table (labels) was populated using 1,000 accounts from the previously gathered data set. Many of the related works, (McCord & Chuah, 2011; Stringhini, et al., 2010), identified that follower/friend ratio was a useful feature. In order to gain a good ratio of bots against humans a preliminary filter was decided upon which filters accounts whom have a follower/friend ratio less than or equal to 0.2. Out of the 485,000 users stored this returned a total of 10,127 users which is far too large. The data set was further reduced by also filtering accounts that have less than 10 friends, this returned a total of 772 accounts. The SQL for this preliminary filter can be found in Appendix A.

Upon first visit to the annotator, a volunteer was required to login. This page made a look up to the volunteers table consisting of an e-mail address, password and last activity timestamp. Based on the provided details the PHP encrypted the password using the MD5 hashing algorithm and looked for a match in the table. Due to the public nature of the annotator, security is key. All PHP code interfaced with the database through the PHP Database Objects[12] (PDO) extension in order to minimise the possibility of SQL injection.

As described earlier, the annotation system required 3 users to label each account. This worked by storing a log of how many times a user has been annotated in the database, along with who made what annotation. When a user visits the annotation page a lookup was made to find all accounts that had been labelled by less than 3 people, this set is ordered randomly using the SQL construct “ORDER BY RAND().” The randomness is added to reduce the possibility, in the case of more than 3 concurrent users, being asked to label the

[12] PHP, n.d. PHP: Introduction - Manual. [Online]
Available at: http://www.php.net/manual/en/intro.pdo.php [Accessed 13/3/2014].

account (thus wasting some of their time as only 3 of the users' annotations would be logged.)

3.2.3 The System in Operation

3.2.3.1 Volunteer gathering

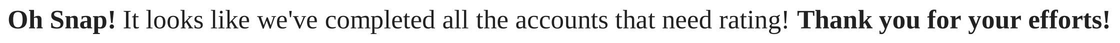
Volunteers were recruited at random by asking members of the public to create an account and classify as many accounts as possible in the time they were able to dedicate. One method of recruitment included posting to Twitter requesting volunteers, this ensured that those who were classifying the accounts were familiar with automated accounts from their daily usage.

In total 28 volunteers were recruited, with each classifying differing volumes of accounts. Two users, including myself, classified the vast majority of the data set forming the first 2 opinions of the inter-rater agreement. The remaining opinion is scattered across the other 26 volunteers who classified between 324 accounts and only 1 account.

3.2.3.2 User Labelling

This section illustrates the system in operation from a volunteer's first visit to the annotation of accounts. A step by step walk through of the system is described below:

1. Appendix C: Login Interface illustrates what was presented to the volunteer upon first visit to the system, requiring a volunteer to authenticate.
2. If a user did not currently have an account they could create one using the "Register" link. Appendix D: Registration Interface illustrates this page. Upon registration the volunteer was immediately authenticated.
    a. The volunteer was then redirected to the annotation interface. A help guide was automatically made aware to the volunteer, illustrated in Appendix E: Help Guide Interface. The automatic display ensured that the reader was aware of the classification procedure and also thanked them for their time.
3. Once a volunteer was authenticated they were presented with the annotation interface to begin annotating accounts, illustrated in Appendix F: Annotation Interface.
4. When a volunteer made a decision about a given account, the form was submitted and another user was immediately displayed.
5. If no user could be found in the accounts to annotate table then the following error message was displayed:



Oh Snap! It looks like we've completed all the accounts that need rating! **Thank you for your efforts!**

3.2.3.3 Inter-rater Agreement

The inter-rater agreement ensures majority agreement across n-volunteers, in this report 3 volunteers rated each twitter account. In order to prove the strength of agreement the Cohen's kappa statistic (Kundell & Polansky., 2003) was employed. Firstly, a PHP program was written to calculate the kappa statistic between two users. The program would perform a number of SQL queries to populate the matrix in Table 3-3:

	Human	**Bot**	**Discard**	**Total**
Human	a	b	c	d
Bot	e	f	g	h
Discard	i	j	k	l
Total	m	n	o	p

Table 3-3 Cohen's Kappa computation matrix

The Kappa statistic is then denoted:

$$\mathrm{K} = \frac{\mathrm{Pr(agree)} - \mathrm{Pr(chance)}}{\mathrm{p} - \mathrm{Pr(chance)}}$$

Where $\mathrm{Pr(agree)}$ is denoted: $\mathrm{Pr(agree)} = \mathrm{a} + \mathrm{b} + \mathrm{c}$ and $\mathrm{Pr(chance)}$ is denoted: $\mathrm{Pr(chance)} = \left(\frac{\mathrm{m} \times \mathrm{d}}{\mathrm{p}}\right) + \left(\frac{\mathrm{n} \times \mathrm{h}}{\mathrm{p}}\right) + \left(\frac{\mathrm{o} \times \mathrm{l}}{\mathrm{p}}\right)$

An algorithm was then used to compare each volunteer against each of the others computing the kappa statistic and the total size of their intersecting sets *T*. The total intersecting set size for the user pair was used to compute the weight, denoted:

$$\mathrm{w} = \frac{\mathrm{T}}{2992}$$

Where 2992 had been pre-calculated to be the sum of all intersection set sizes across user pairs. Finally, the average weighted kappa is thus denoted:

$$\mathrm{K_w} = (\mathrm{w_1} \times \mathrm{k_1}) + (\mathrm{w_2} \times \mathrm{k_2}) + \ldots (\mathrm{w_n} \times \mathrm{k_n})$$

The average weighted kappa in this instance had been computed to 0.637 rounded to 3 decimal places. In (Kundell & Polansky., 2003) guidelines were presented for analysing the strength of agreement indicated by K values. From Table 3-4 it is evident that there was substantial agreement between the 28 volunteers used for the classification of accounts. This result suggested that there was accuracy in the pre-classified data set and hence should be reliable for the machine learning classifiers used in section 4.

K	**Strength of Agreement beyond Chance**
<0	Poor
0 - 0.2	Slight
0.21 - 0.4	Fair
0.41 - 0.6	Moderate
0.61 - 0.8	Substantial
0.81 - 1	Almost perfect

Table 3-4 Inter-rater agreement levels

4 ACCOUNT CLASSIFICATION

Machine learning is a branch of artificial intelligence providing the ability for computers to learn from a given data set without having to be hard-coded. In this report classification models are used to automatically determine whether any given Twitter account is a bot or human. This section will detail the features employed, the various steps for setting up the classifiers and finally demonstrate the experimentation results.

4.1 Features

In machine learning, features are individual measurable variables for a given observed property in the data set, for example: does the animal have fur. Features are typically of numeric type but can also be structural such as strings or graphs. This report categorises the features to be classified into state-of-the-art features (those indicated to be promising in previous research), and new features (that have not been implemented in the past).

4.1.1 State of the Art Features

4.1.1.1 Number of Followers

The number of followers was used by (Wang, 2010). During the data gathering stage this was collected from the *user_timeline* API as a numeric value.

4.1.1.2 Number of Friends

The number of friends was employed by (Benevenuto, et al., 2010; Chu, et al., 2012; Stringhini, et al., 2010; McCord & Chuah, 2011; Wang, 2010). This value was also collected during the data gathering stage from the *user_timeline* API as a numeric value.

4.1.1.3 Follower to Friend ratio

Follower to friend ratio was commonly used across the related works, it was computed based on the number of followers and the number of friends described above. Let $\mathrm{U_{fol}}$denote the number of followers a user has, $\mathrm{U_{fr}}$ denote the number of friends a user has and $\mathrm{r_{ff}}$ denote the ratio. The follower ratio was thus denoted:

$$\mathrm{r_{ff}} = \frac{\mathrm{U_{fol}}}{\mathrm{U_{fr}}}$$

In the cases where the number of friends $\mathrm{U_{fr}}$ a user had is zero, $\mathrm{r_{ff}}$ was automatically set to 0. The result $\mathrm{r_{ff}}$ was always rounded to 2 decimal places.

4.1.1.4 Tweet Source

(Chu, et al., 2012) identified tweet source as being one of the most reliable features in identifying bots on Twitter. This feature was implemented by finding the most frequent source across all of a user's tweets. The result was stored as a nominal data type specifying the most frequent source for that particular user.

4.1.1.5 Mean URLs

A number of research efforts used the number of URLs that a user has tweeted, promising results were identified from (Chu, et al., 2012; Chu, et al., 2012; Stringhini, et al., 2010; McCord & Chuah, 2011). This feature was implemented as the average number of URLs across all of a user’s tweets. Let *u* denote a URL and *c* denote the total number of tweets processed, ignoring those that are identified as retweets; tweets are identified as retweets when prefixed by “RT” or stated so by the API. The mean number of URLs μ_u was denoted:

$$\mu_u = \frac{\Sigma\ \mathrm{u}}{\mathrm{c}}$$

URLs were identified within a tweet by splitting the tweet into tokens based on whitespace separation. A URL was then validated against the MalformedURLException class. The result of the mean was rounded to 2 decimal places for consistency across features.

4.1.1.6 Mean Hashtags

(Benevenuto, et al., 2010; Stringhini, et al., 2010; McCord & Chuah, 2011) employed the number of hashtags as a feature, again with promising results. This feature was implemented as the average number of hashtags across a user's set of tweets. Let *h* denote a hashtag and *c* denote the total number of tweets processed, ignoring retweets. The mean μ_h was thus calculated as:

$$\mu_h = \frac{\Sigma\ \mathrm{h}}{\mathrm{c}}$$

Hashtags were identified within a tweet by splitting the tweet into tokens based on whitespace. Each token was then matched against the regular expression, _((?<!\w)#\w+[^\s]*),_ specifying that a hashtag should begin with a # and immediately be followed by 1 or more word characters up until white space. The result of mean was again rounded to 2 decimal places and stored as a numeric value.

4.1.1.7 Mean Characters

(Benevenuto, et al., 2010) looked at the number of characters per tweet, (McCord & Chuah, 2011) also look at the total length of a tweet. This feature was implemented as the mean number of characters across a user’s tweets. Let *l* denote the total number of characters for a given tweet and *c* denote the total number of tweets processed, ignoring retweet. The mean μ_c was denoted:

$$\mu_c = \frac{\Sigma \mathrm{l}}{\mathrm{c}}$$

4.1.1.8 Mean Words

Similarly to the mean number of characters, (Benevenuto, et al., 2010) also used the number of words per tweet. This feature was implemented as the mean number of words per tweet, across the user’s entire set of tweets. A word was identified by trimming the tweet of excess white space and then splitting the tweet into individual tokens. Let *w* denote the total number of words in a tweet and *c* denote the total number of tweets processed, ignoring retweets. The mean μ_w was calculated as:

$$\mu_w = \frac{\Sigma\ \mathrm{w}}{\mathrm{c}}$$

4.1.1.9 Total duplicate tweets

(Wang, 2010; Chu, et al., 2012; Stringhini, et al., 2010; Gianvecchio, et al., 2011) identified that a number of spam accounts typically tweet identical contents over a period of time. This feature was implemented using the Levenshtein distance algorithm, similarly to (Wang, 2010)’s approach. The Levenshtein distance is defined as the minimum cost of transforming one string into another, the result is the total number of characters that differ in the transformation. The approach ignored any tweets identified as being retweeted as done in previously described features. For every tweet *t1* it was compared against the rest of a user's tweets, when a tweet has been processed it was removed from the set to ensure it was not processed again. Whenever a distance of zero was computed, the total number of duplicates *d* is incremented. To ensure that the total was given on the same scale no matter how many tweets a user had in total, the result t_d was converted to $\log_{10}$, denoted:

$$t_d = \log_{10}(\Sigma\ \mathrm{d})$$

4.1.1.10 Total interactions

The number of times a user interacted with others on the social network was a feature common in the work of (Wang, 2010; McCord & Chuah, 2011). This feature was implemented by checking for every tweet, that was not a retweet, did the user mention/reply to another user. User interactions denoted as *i* were detected within tweets using regular expressions:

```
(?<=^|(?<=[^a-zA-Z0-9-\.]))@([A-Za-z_]+[A-Za-z0-9_]+)
```

The regular expression specified that a username must be prefixed by an @ symbol and immediately followed by an alpha character, the final part of the username can be complete with alphanumeric characters. The expression also ensured that any characters before the @ are outside of alphanumeric range, allowing for symbols before the @.

Similarly to the total number of duplicate tweets the result t_i was converted to $\log_{10}$, denoted:

$$t_i = \log_{10}(\Sigma\ \mathrm{i})$$

4.1.1.11 Total retweets

The number of retweets was used by (McCord & Chuah, 2011). This feature was implemented by checking for every tweet *t* had the API marked it as retweeted or was the tweet prefixed by "RT", if it was *r* was incremented. The total t_r was again converted to $\log_{10}$, denoted:

$$t_r = \log_{10}(\Sigma\ \mathrm{r})$$

4.1.2 New Features

New features are those which no prior research efforts have touched upon, which will be introduced in this report. The related work section had indicated that no research had looked into natural language processing techniques, leading this research to consider this as a feature. I was led to look at this as a feature as it was indicated by those who had taken part in the annotation process. The volunteers were e-mailed asking for any noticeable factors leading them to choose between a bot or human. Feedback from this brief survey revealed a number of interesting characteristics for example:

- *"Bots always tweet about the same thing?"*
- *"Bots often exhibit fixed patterns in their spelling/grammar whereas humans are less predictable."*
- *"Bots typically showed no emotion, whilst a human may have a variety of positive/negative tweets."*
- *"Tweets from bots often contained a link / URL, sometimes this was duplicated"*

The results from this short, informal survey revealed great promise in the belief that natural language processing could offer significant improvements on previous research efforts. Based on these suggestions, six new features were implemented 4 of which can be considered natural language processing (mean and standard deviation of tweet sentiment, hashtag context and spelling) with two being extensions upon previously researched features (use of duplicate URLs and default profile picture).

4.1.2.1 Sentiment

The survey revealed that a number of volunteers believed bots exhibited regular patterns in the way that they communicate, as previously indicated by (Gianvecchio, et al., 2011). Some volunteers believed that this pattern was particularly evident in the case of emotion. For example, humans would have a mixture of positive and negative emotion across their tweets whilst bots are typically consistently negative for example publishing news articles.

The sentiment feature set was implemented using the Stanford CoreNLP (University of Stanford, n.d.) third-party Java library. Stanford CoreNLP is a natural language processing framework providing ease in implementing language analysis of text. Prior to version 3.3.0 of the framework sentiment analysis had not been included, with the current version being 3.3.1 (introduced in 2014), making this feature very cutting edge research.

To implement sentiment analysis the third-party library must first be imported into the java class path. To implement the sentiment analyser firstly a class was created taking a string of text (the tweet) into the constructor. Once the instance had been initialised a separate method performed the analysis. This method first created a new instance of StanfordCoreNLP specifying the properties to be used, here it was specified that sentiment analysis was to be used and the variable ‘ssplit.isOneSentence’ was set to true to ensure any new lines/whitespace in the tweet did not break the analysis early. Next, the CoreAnnotations.SentencesAnnotation class was used to produce a parse tree of the tweet, the result being stored as a CoreMap. The SentimentCoreAnnotations.AnnotatedTree class was then used to produce a tree identifying the sentiment for each token of the string. Finally, the RNNCoreAnntoations class was used to calculate the root sentiment of the tree returning the sentiment class as an integer. This integer can be one of 5: 0 (very negative), 1 (negative), 2 (neutral), 3 (positive) and 4 (very positive.)

The sentiment feature set was broken down into two separate calculations:

1. Mean sentiment
   The mean was calculated by iterating over all a given users tweets, any tweets that were identified as retweeted by the API or any tweets that began with the term “RT” were automatically ignored. The tweets that weren’t ignored, are passed into the analyser class to compute the sentiment score *s*, which was then incremented to the total *t*. A log of how many tweets were processed was also incremented, *c*. The mean, μ_{se}, is denoted:

   $$\mu_{se} = \frac{\Sigma s}{\mathrm{c}}$$

   The result of this being rounded to the nearest whole number (a sentiment score is a whole number between 0 and 4.)

2. Standard deviation of the sentiment
   The standard deviation of the sentiment was calculated based on the previously determined mean for the given user. For each tweet of the user,

given the mean, *m*, and the sentiment of the tweet, *s*, a count, *c*, of how many tweets have been iterated was stored along with the sum calculation between *s* and *m*. One would anticipate that a low standard deviation would indicate a predictable pattern in an account sentiment and a high value would be less predictable, likely the case for humans. The standard deviation σ was rounded to two decimal places and is denoted:

$$\sigma = \sqrt{\frac{\Sigma(s - m)^2}{c}}$$

4.1.2.2 HashTag context

(Benevenuto, et al., 2010) identify spammers who are aggressive in their strategy by posting malicious contents into trending topics. A quick study of the pre-classified data set and browse through the current Twitter trending topics revealed that there are a number of users who tweet unrelated contents to the given topic. This presented the idea of identifying the context of a tweet and comparing this in relation to the trending topic term. For example: given the tweet "*Cheap viagra 70% off sale, follow back for details!! #WorldCup*" it is clearly evident that this is completely unrelated to the #WorldCup football trending topic.

Previous work such as that from (Stringhini, et al., 2010) and (Chu, et al., 2012) use language analysis to find semantic similar of tweets. Building on this, this feature is computed as follows: let *h* denote a hashtag in a tweet and *t* denote a term within a tweet. The similarity was then computed for every hashtag, *h*, against every term, *t*, the result of this was averaged to produce the mean similarity for that particular tweet. This could then be further averaged out across all of the users' tweets. The average tweet context against hashtags μ_{hc} across all of a user's tweets is thus denoted as:

$$\mu_{hc} = \frac{\Sigma\left(\Sigma s \div (\Sigma t + \Sigma h)\right)}{c}$$

One would anticipate this would produce a low similarity if the hashtags do not match the content and a high similarity if they do match the contents.

Before the similarity of terms and hashtags within a tweet could be computed pre-processing must occur to disregard unnecessary information. In order to build up an accurate profile of the user, any retweets were identified by tweets prefixed with "RT" or where the API had previously specified it as retweeted and as such disregarded. The tweet was then split into individual terms by separating based on whitespace. Any term in the set that was identified as a username or URL was removed from the set. If the term was prefixed by a hash character it was identified as a hashtag and stored in a set, with all other terms being stored in a separate set; in both cases any punctuation was removed. Testing of this pre-processing revealed that a number of trending topics were composed of camel cases (multiple words separated by capital.) This posed the problem that these words could not be found within the WordNet database. In the occasion where camel cases were identified these were separated into individual hashtags.

The similarity analyser is implemented using WordNet (Fellbaum, 1999) based measures. In order to interface with the WordNet database three third-party Java libraries are used, these being the MIT Java Interface to WordNet (Finlayson, 2013) and WordNet similarity for Java[13] (*ws4j*); a requirement of this library is that JawJaw[14] must also be included in the class path. Once the necessary libraries have been imported an analyser class was created to compute the similarity between two words. Firstly, WordNet is organised by synonym sets so (Finlayson, 2013) was used to find the root meaning of the provided word. Secondly, the implementation of the Resnik similarity measure provided by *ws4j* is initialised. Using the RelatednessCalculator class provided by *ws4j*, POS pairs were iterated over to find whether a word was a verb, noun, or adverb. Once the word had been found a synonym set was returned as a list for each word being compared. The synonym sets were then iterated over comparing the relatedness of each word in the set. On each iteration the score was compared against the previous until a maximum was found.

4.1.2.3 Spelling errors

Feedback from the survey of participating volunteers had revealed that bots exhibited regular patterns in their spelling in that they were typically grammatically correct and had no spelling errors. In comparison, humans would use abbreviations and typically be much less grammatically correct thus less formal in their communications.

This feature calculated the mean number of spelling errors across a user's set of tweets. Let *t* denote a tweet and *sp* denote the number of spelling errors found for *t*. The sum of *sp* was calculated across all of a user's tweets and then divided by the total number of tweets that were processed *c*. Thus the mean μ_{spl} was denoted:

$$\mu_{spl} = \frac{\Sigma \mathrm{sp}}{\mathrm{c}}$$

The result was rounded to two decimal places for consistency with other features.

The spelling analyser was implemented using a third-party Java library known as Jazzy[15] which analyses bodies of text for spelling and grammatical errors. In order to interface with Jazzy it must first be included into the Java class path, as with all other external libraries. Prior to passing any data into the library the tweets are pre-processed. Similarly to the hashtag contexts, any tweets that are retweeted are disregarded, likewise usernames, hashtags and incomplete URLs are stripped from the tweet. Testing revealed that a number of words such as “tweet” weren’t recognised by the dictionary so these were also removed. Finally using the default English dictionary provided by Jazzy the analyser initialises a hash map of the dictionary and new instance of the spell checker. The tweet is then simply passed into the checkSpelling() method which returns the number of errors in the tweet.

4.1.2.4 Duplicate URLs

To further the work of (Benevenuto, et al., 2010; Chu, et al., 2012; McCord & Chuah, 2011) who count the number of URLs per tweet and also check the URL against blacklists, it was believed that duplicate URLs could be a key factor in determining bots. Looking through the pre-classified data set it was evident that a number of accounts did post duplicate URLs particularly in the case of updates from gaming apps.

[13] Shima, H., 2013. ws4j - WordNet Similarity for Java - Google Project Hosting. [Online] Available at: https://code.google.com/p/ws4j/ [Accessed 15 3 2014].

[14] Shima, H., n.d. jawjaw - Java Wrapper for Japanese WordNet - Google Project Hosting. [Online] Available at: https://code.google.com/p/jawjaw/ [Accessed 15 3 2014].

[15] Idzelis, M., n.d. The Java Open Source Spell Checker. [Online] Available at: http://jazzy.sourceforge.net/ [Accessed 15 3 2014].

Detecting URLs was implemented similarly to the above described features in that tweets detecting as retweets were disregarded. The tweet was then split into its individual tokens separated by whitespace. Each token was then checked to see if it was a valid URL using the MalformedURLException class, and if it was it was added to set. Further iterations then checked to see whether the valid URL token already existed within the set, if so the total count *t* was incremented. Finally, t_u was converted and stored as $\log_{10}$to represent the total on the same scale regardless of the number of tweets per user:

$$\mathrm{t}_u = \log_{10}(\Sigma\ \mathrm{t})$$

4.1.2.5 Default profile picture

(Wang, 2010) indicated that the data gathering process was performed against users who have a custom user icon. Quick analysis of the pre-classified data set revealed that a number of the accounts that had been identified as bots had the default profile picture set, whilst the vast majority of humans had set it to something personal to them.

During the data gathering stage the *user_timeline* API returns whether a given account at that moment in time has the default profile icon currently set, this was returned as a boolean value. Due to this value already being stored in the data set it was simply set as a binary feature whereby 1 indicates that an account does have the default profile picture and 0 does not.

4.2 Implementation

The implementation of the classification section of this report made use of the Weka machine learning tool set (Hall, et al., 2009). Weka is a collection of pre-built machine learning algorithms, providing a framework for classification, regression, clustering and visualization. Amongst the provided Java API, Weka also provided a native client for ease of use in small-medium scale machine learning tasks. This report made use of the native client for all machine learning processes.

4.2.1 Constructing the ARFF file

The Weka client required the construction of an ARFF file (Attribute Relation File-Format.) The ARFF file is an ASCII text file describing the list of instances sharing a set of attributes/features. The file is composed of a header describing the set of features and a body containing comma delimited data, known in Weka as instances where each new line marks the start of a new instance. In this implementation the ARFF file was produced automatically based on an interface to the database housing the pre-classified data and a configuration file which stated which attributes to include in a given instance.

Appendix G: Class Diagram illustrates that the implementation was composed of 3 key parts: getting the pre-classified data from the database, pre-processing the data into the features described in section 4.1 and then finally outputting the internal Java structures into the appropriate ARFF file format.

The database handler makes use of the MySQL JDBC[16] third-party library to access the pre-classified data on a separate, remote, machine. As with the earlier described libraries, this must also be imported into the class path. Once imported, the set up required that the database access details be provided to the DriverManager class. If the driver was able to successfully connect, a query was made to the database to gather all of the pre-classified data including both user details and tweets. Each row of the returned ResultSet was then iterated by creating a User object which housed account details and also an array of Tweet objects for each of the users tweets.

The pre-processor contains a list of static methods for computing the features described in section 4.1. The first step towards building the ARFF file involved computing the values and storing these in a Java data structure. The chosen data structure was an ArrayList of maps, where each item in the ArrayList was an instance or one row of individual user data. A TreeMap with an Integer key and Object value was used for each item. The integer key corresponds to the ID representing the attribute from the configuration file. For example, at ID 0 existed the total number of followers for each user. The configuration file contained a list of Weka Attribute classes, specifying whether the data type is numeric or nominal. All attributes had a numeric value except for the tweet source and class which were of nominal type.

The final stage involved converting the ArrayList of TreeMaps into the required format for the ARFF file. The implementation of this involved the creation of a custom class known as the “ArffBuilder”. This class takes the attributes specified in the configuration file and first build a FastVector consisting of each attribute to build the header for the ARFF file. Secondly, for each item of the ArrayList the TreeMap would be found. Furthermore, the

[16] MySQL, n.d. MySQL :: Download Connector/J. [Online]
Available at: http://dev.mysql.com/downloads/connector/j/ [Accessed 15/3/2014].

TreeMap would then be iterated over using the integer key to find the associated attribute in the configuration file. Once the attribute had been found it would determine whether it is of numeric or nominal type. If the value was nominal the value would simply be stored in a result array of doubles, otherwise the list of nominal types specified in the configuration file would be retrieved. The returned nominal types would then be searched to find the value at the current position in the TreeMap. For each iteration the resulting double array would be stored in an array of instances. Upon completion of iteration through the ArrayList the instances variable would be returned and printed to console. Weka nicely provided a toString() method for formatting the Instances object into the requirement format for the ARFF file.

4.2.2 Using Weka Explorer

The Weka Explorer is included as part of the Weka tool set (Hall, et al., 2009). The program provides ease of use in running machine learning algorithms such as classifiers, clustering and visualization on a given ARFF file. Once the ARFF file had been computed it was very easy to begin experimenting. The “Preprocess” section allowed one to import the ARFF file and select which features are to be used in later machine learning analysis.

The “Classify” section of the Weka Explorer could then be used to apply classification algorithms to the provided data set. The Weka Explorer automatically disables classifies that are unavailable based on the type of data imported. In this report Bayes Net, Naive Bayes, Random Forest and SMO classifiers are employed to compare their differences in performance. Previous research efforts such as (Chu, et al., 2012) and (Benevenuto, et al., 2010) have solely focused on one particular type of classifier; it is possible that others may perform better under certain feature sets.

In this report all the classifiers are computed using 10-fold cross validation. In each test case, the sample is partitioned into 10 equal size subsamples. Of the 10 subsamples, 1 was retained as validation data for testing the classifier and the remaining 9 were used as training data. The cross-validation process was then repeated 10 times with each subsample being used exactly once as the testing data.

Once the 10-fold cross validation process had completed, Weka Explorer outputs the results of the classifier detailing accuracy by class (Human/Bot). For evaluating the accuracy of the classifier, the weighted F-measure was used. The F-measure is a standard statistical test for binary classification systems, combining both the precision and recall with equal weights. Given the confusion matrix in Table 4-1:

		Condition (as determined by Gold Standard)	
		Bot	Human
Test outcome	Bot	a	b
	Human	c	d

Table 4-1 F Measure confusion matrix

The F-measure is denoted as:

$$\mathrm{F}_1 = 2 \times \left(\frac{\mathrm{P} \times \mathrm{R}}{\mathrm{P} + \mathrm{R}}\right)$$

Where the precision is denoted $P = \frac{a}{(a+c)}$, and the recall is $R = \frac{a}{(a+b)}$.

4.3 Experimentation Results

This section summarises the results of each classifier under a given set of features. The features have been subdivided for comparison, into distinct sets these being: state-of-the-art, user-based, content-based and new features. The section will finish with all of the features combined and an optimal feature set having removed any that are poorly performing.

4.3.1 State-of-the-art vs New Features

The state-of-the-art feature set included those eleven features that previous researchers had identified to return promising results, a list of these features can be found in section 4.1.1. In comparison, the new feature set included those six features that have been presented in this report, described in section 4.1.2.

		F-Measure	
		State-of-the-art Features	New Features
Classifier	Bayes Net	0.915	0.848
	Naïve Bayes	0.88	0.846
	Random Forest	0.922	0.819
	SMO	0.913	0.845

Table 4-2 State-of-the-art performance vs. New Features

Table 4-2 shows the performance of the state-of-the-art feature set against those introduced in this report. Initially inspection of the results showed that there had been a clear decline in the F-measure for those newly introduced. This being said, (Benevenuto, et al., 2010; Chu, et al., 2012) identified that smaller set of features can limit the accuracy as in the case of those newly introduced.

In the case of the state-of-the-art features Random Forest had the highest accuracy at 92.2%, whilst Naïve Bayes was much further behind on 88%. Random Forest is an ensemble classifier that combines the decisions together of several classification models, thus making it generally more robust to noise. In comparison, Naïve Bayes is a simple probabilistic classifier. The extreme difference between these two F-measures could suggest that the state-of-the-art features carry a lot of noise.

The features introduced in this report had much lower accuracies than those seen in the state-of-the-art. In this occasion, all classifiers had very similar accuracies however Bayes Net and Naïve Bayes exhibited the highest. In comparison to the state-of-the-art Random Forest exhibited the lowest accuracy. This could suggest that the newly introduced features were much less noisy and were truly independent on one another; Naïve Bayes makes the naive assumption of independence between feature pairs.

4.3.2 User-based Features vs. Content-based Features

User-based features consist of those that were limited to the properties of an account; this feature set included a mixture of those that were state-of-the-art and those that were being introduced. Out of the described features in section 4.1, the following were identified as user-based features:

- default profile picture
- number of followers
- number of friends
- follower to friend ratio

Content-based features consist of those that are reliant on content provided by the user, thus tweets; again, this feature set includes a mixture of those that were state-of-the-art and those that were being introduced. Out of the described features in section 4.1, the following were identified as content-based features:

- duplicate URLs,
- source of tweets,
- mean URLs,
- mean hashtags,
- mean characters,
- mean words,
- total duplicate tweet,
- total interactions,
- total retweets,
- sentiment,
- hashtag context,
- spelling errors

		F-Measure	
		User-based Features	Content-based Features
Classifier	Bayes Net	0.835	0.925
	Naïve Bayes	0.471	0.919
	Random Forest	0.808	0.929
	SMO	0.482	0.934

Table 4-3 User-based feature performance vs. Content-based features

Table 4-3 compared the performance of user-based features against content-based features. Initially it was clear that content-based features performed much better than user-based, with accuracies upwards of 90% whilst user-based features fell as low as 47%. Interestingly, in the case of user-based features Bayes Net and Random Forest had an accuracy almost 30% higher than SMO/Naïve Bayes. As mentioned in section 4.3.1 the poor accuracy in this occasion could be put down to large amounts of noise in the data set which would explain the improved accuracy of Random Forest/Bayes Net.

In the case of content-based features, SMO exhibited the highest accuracy of 93.4%, with Random Forest not far behind on 92.9%. These results could suggest that content-based features provide higher information gain to the classifier than those of user-based features and were thus more powerful. This being said, the twelve content-based features against 4 user-based features could have affected the result in that much more information is contributed in the case of content-based features.

4.3.3 Combined Features

The combined feature set combined all those described in both new and state-of-the-art features sets, a list of which can be found in section 4.1.1 and 4.1.2.

Classifier	F-Measure
Bayes Net	0.926
Naïve Bayes	0.905
Random Forest	0.924
SMO	0.93

Table 4-4 Performance of all features combined

Table 4-4 showed that SMO yielded the highest accuracy when all of the proposed features from section 4.1 were combined. Interestingly, when this is compared against the results presented in Table 4-3 the accuracy of all results has decreased except for Bayes Net where there had been a marginal increase. In case of the SMO, that was previously the most accurate for content-based features, the accuracy has decreased by 0.04%. This confirmed that user-based features consisted of a lot of noise. Likewise, more features doesn't necessarily have a positive effect on the accuracy.

When comparing the combined feature set against that of the state-of-the-art in Table 4-2 it was noted that there had been a clear increase in the accuracy, despite potentially noisy user-based content. In the case of Bayes Net, Naïve Bayes and SMO there had been an increase on the state-of-the-art of approximately ~1%. This increase suggested that the newly introduced features were of high information gain to the classifier and hence having a positive effect on the accuracy. Similarly, in the case of Random Forest a marginal increase was seen but suggested that noise in the state-of-the-art features was limiting the growth of the accuracy.

4.3.4 Best Model Selection

In order to calculate an optimal feature set, the "Select Attributes" section of the Weka Explorer was used. This section of the Weka tool set allowed me to evaluate attribution selection against various algorithms. In this implementation the InfoGainAttributeEval algorithm was used to evaluate the worth of each attribute by measuring the information gain with respect to the class (Human/Bot). The search method was then set to Ranker to order attributes by their individual evaluations.

The results of ranking the attributes by information gain are indicated in Table 4-5. It was most evident that the tweet source contributed highly to classifying accounts as Humans/Bots. Throughout the pre-classification this became clear with a number of volunteers suggesting that bots would typically post from applications, whilst humans would be consistently from web/mobile devices. (Chu, et al., 2012) also state that the tweet source contributes the most information gain towards the classifiers. The rest of the results sat around the 0.4 range, all of which of are numeric values as opposed to the nominal type for the tweet source. For the 5 numeric types, box plots are illustrated in appendix B highlighting the difference in values between both classes (Human/Bots).

Table 4-5 shows that the total number of user interactions is the second highest attribute for information gain. Appendix B.4 illustrates the sample of results for each user comparing those classified as humans against bots. The box plot clearly illustrates that bots typically interact much less with other users than humans, with a noticeable crossover occurring above 1.0. A possible reason for this crossover is that some automated accounts are malicious and deliberately target users for spam. In this instance multiple automated accounts were used to mention the same user thus spamming the target with notifications.

Information Gain	**Attribute**
0.7136	Tweet Source
0.4331	Total number of interactions
0.4296	Mean URLs
0.4109	Total Retweets
0.4076	Total friends
0.4009	Standard deviation of the sentiment

Table 4-5 Attributed ranked by information gain

Appendix B.1 illustrates the mean number of URLs for both humans and bots. It is clearly evident that bots typically post many more URLs than humans. It is notable that in the case of humans there are extreme cases where a number of accounts are identified as posting a large amount of URLs.

Appendix B.5 illustrates the total number of retweets. From the graph it is evident that around 50% of a humans tweets are retweets, though across the data set there are cases with zero retweets. In the case of bots an interesting case is presented with a large proportion of extreme cases with large numbers of retweets. However, the majority case for bots is that they typically do not retweet other user content. The tweet-source identifies this in that a large proportion of the accounts identified as bots simply scrape content from the web as opposed to searching the social network for data to retweet.

Appendix B.3 shows the number of friends across humans and bots. The graph presents the case that bots typically do not have any friends, though there are extreme cases where they have a large proportion. This is typical in the case where the account is legitimately identified as a bot, and has a useful purpose. On the other hand, it is evident that humans typically have a number of friends in the range of a couple of thousand; as one would expect there are extreme cases for popular accounts.

Appendix B.2 illustrates the newly implemented standard deviation of the sentiment. The graph illustrates a very clear separation between bots and humans that has not been seen in the previous illustrations. Bots typically have a much lower sentiment suggesting that they're consistent, whilst humans have a slightly higher value indicating less-predictability.

The optimal feature set was decided upon by removing those attributes that have a ranking below 0.1. The values that were thus removed include: total duplicate tweet, total followers, total duplicate URLs, hashtag context and default profile picture. The remaining features were included in the optimal feature set, these are detailed below ordered by information gain:

1. Tweet source
2. Total interactions
3. Mean number of URLs
4. Total retweets
7. Follower to friend ratio
8. Mean number of hashtags
9. Mean number of spelling errors
10. Mean number of characters
11. Mean sentiment of tweets

5. Standard deviation of tweet sentiment
6. Mean number of words

Classifier	F-Measure
Bayes Net	0.927
Naïve Bayes	0.926
Random Forest	0.94
SMO	0.931

Table 4-6 Optimal feature set performance

Table 4-6 shows that there is a clear increase in the accuracy of the classifiers when removing those believed to provide low information gain. When these results were compared to Table 4-4, an increase was seen across all classifiers. The most notable increases were seen in the case of Naïve Bayes and Random Forest, with only marginal increases of 0.01% for Bayes Net and SMO.

From Table 4-6 it was clear that Naïve Bayes had seen a steady increase was seen across sections 4.3.1 – 4.3.3. Originally in the state-of-the-art Naïve Bayes had an accuracy of 88%, this increased to 90.5% when the features were combined. Once the noisy or features of low information gain were removed the highest accuracy was seen, improving upon that of (Wang, 2010). Interestingly, Naive Bayes sees a further increase when removing the next lowest information gain attribute, mean number of words, increasing to a value 0.934. However, removing this feature had a knock on effect on other classifiers seeing a decrease in the result. If further features were removed (information gain >0.1) the F-measure scores would gradually decline.

Across sections 4.3.1 to 4.3.3 Random Forest had typically performed best or close to best. In the optimal feature set an each an increase of 0.16 was seen from the results presented in Table 4-4. This result was a clear improvement upon the state-of-the-art and confirms that features indicating account patterns such as the standard deviation of tweet sentiment contribute highly to the information gain of a classifier. This result was an improvement upon those seen in the earlier research of (Wang, 2010) and (Tavares & Faisal, 2013) who both try to detect bots on Twitter.

5 CONCLUSION

5.1 Review of the Aims

This report began with the aim of detecting bots on Twitter using machine learning approaches. It was initially hoped that an existing data set would be available, however it became apparent that a number of existing data sets were not suitable for the scope of this study. The report progressed to build a large data set from scratch including Twitter account details and 200 of their most recent tweets.

An annotation system was produced to allow members of the public to classify accounts as bots or humans. Here an inter-rater agreement of 3 users was found to improve upon previously built data sets, likewise Cohen’s Kappa was used to prove the strength of inter-rater agreement. Throughout the data gathering process, volunteers posed questions that led to the introduction of new features presented in this report. If this process were to be repeated, a larger set of volunteers would help to diversify those annotating the accounts but also overcome the fact that in this instance, 2 users labelled the majority of the data set, which could affect the inter-rater agreement.

New and state-of-the-art features are compared in this report against four traditional classifiers. The results yielded from this study are an improvement upon early research in this field, but they also have similarities to recent efforts. Natural language processing features were introduced with the standard deviation of tweet sentiment proving to be of high information gain to the classifiers. That being said, a number of other natural language processing techniques were also implemented in the form of the mean across a user’s tweets. It is identified that the standard deviation of features proves to be of higher information gain than using the mean.

5.2 Future Work

It is hoped that future work in this field would improve upon the analysis of tweet context against hashtag terms. A number of problems were presented when processing this feature in that hashtags terms do not typically exist within dictionaries, for example in the case of multiple words joined together in lower case. Similarly, (Tavares & Faisal, 2013) managed to successfully detect bots based solely on ‘timing entropy’. Given more time, this report would look to implementing an entropy based component for the timing and prediction of tweets to help further build a profile of a user and their predictable, or not so predictable, actions.

(Chu, et al., 2012; Tavares & Faisal, 2013) states that there are multiple types of bot, therefore I envisage future work would look to characterise those accounts detected as bots into distinct categories. In the proposal bot taxonomies had been identified, future work in this field may look to categorise accounts detected as bots as it is possible that a number of accounts could be a mix of human/bot (cyborg). Furthermore, the pre-classified data set could be improved upon by increasing the rate of agreement between volunteers thus improving the overall reliability of the results.

5.3 Closing Comments

From this report I have learnt to use statistical techniques such as Cohen’s Kappa to prove the strength of agreement between multiple user pairs. Also, how to deploy machine learning classifiers for training against a data set using the Weka native framework. Likewise, the effects that different numbers and types of features can have on the accuracy of a classifier. In most occasions the Random Forest classifier proved most resilient to noise often yielding the highest result.

To conclude, this report has built upon previous research efforts of others, whilst contributing the introduction of natural language processing based features. A data set of 1,000 users is made available to the community that has proven substantial inter-rater agreement, in the identification of bots and humans on Twitter. A classifier is then presented implementing new and state-of-the-art features yielding slightly better results than previously seen in similar research efforts. The information gain is also contributed for the feature set indicating which features are best in this particular implementation.

6 ACKNOWLEDGEMENTS

I sincerely thank Matthew Rowe for his insightful comments and suggestions.

REFERENCES

Benevenuto, F., Magno, G., Rodrigues, T. & Almeida, V., 2010. *Detecting spammers on twitter.* s.l., s.n., p. 12.

Chu, Z., Gianvecchio, S., Wang, H. & Jajodia, S., 2012. Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg?. *Dependable and Secure Computing, IEEE Transactions on,* 9(6), pp. 811-824.

Chu, Z., Widjaja, I. & Wang, H., 2012. *Detecting social spam campaigns on twitter.* s.l., s.n., pp. 455-472.

Fellbaum, C., 1999. *WordNet.* s.l.:Wiley Online Library.

Finlayson, M. A., 2013. Code for Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation.

Gianvecchio, S., Xie, M., Wu, Z. & Wang, H., 2011. Humans and bots in internet chat: measurement, analysis, and automated classification. *IEEE/ACM Transactions on Networking (TON),* 19(5), pp. 1557-1571.

Hall, M. et al., 2009. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter,* 11(1), pp. 10-18.

Kundell, H. & Polansky., M., 2003. Measurement of observer agreement. *Radiology,* Volume 228, pp. 303-308.

McCord, M. & Chuah, M., 2011. Spam detection on twitter using traditional classifiers. In: *Autonomic and Trusted Computing.* s.l.:Springer, pp. 175-186.

Statistic Brain, 2014. *Twitter Statistics - Statistic Brain.* [Online]
Available at: http://www.statisticbrain.com/twitter-statistics/
[Accessed 18 3 2014].

Stringhini, G., Kruegel, C. & Vigna, G., 2010. *Detecting spammers on social networks.* s.l., s.n., pp. 1-9.

Tavares, G. & Faisal, A., 2013. Scaling-laws of human broadcast communication enable distinction between human, corporate and robot twitter users. *PloS one,* 8(7), p. e65774.

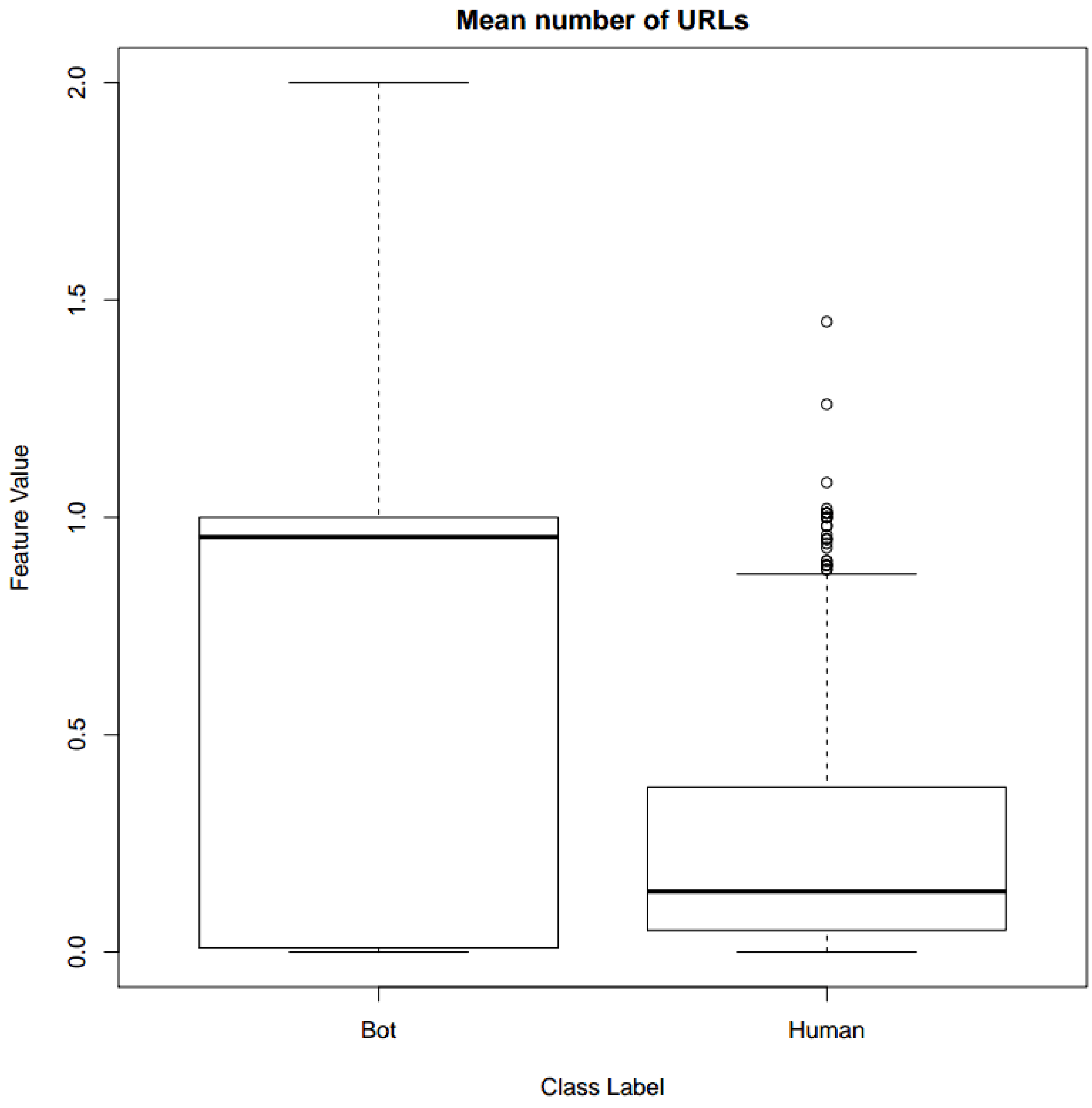
University of Stanford, n.d. *The Stanford NLP (Natural Language Processing) Group.* [Online]
Available at: http://nlp.stanford.edu/software/corenlp.shtml#sentiment
[Accessed 15 3 2014].

Wang, A. H., 2010. Detecting spam bots in online social networking sites: a machine learning approach. In: *Data and Applications Security and Privacy XXIV.* s.l.:Springer, pp. 335-342.

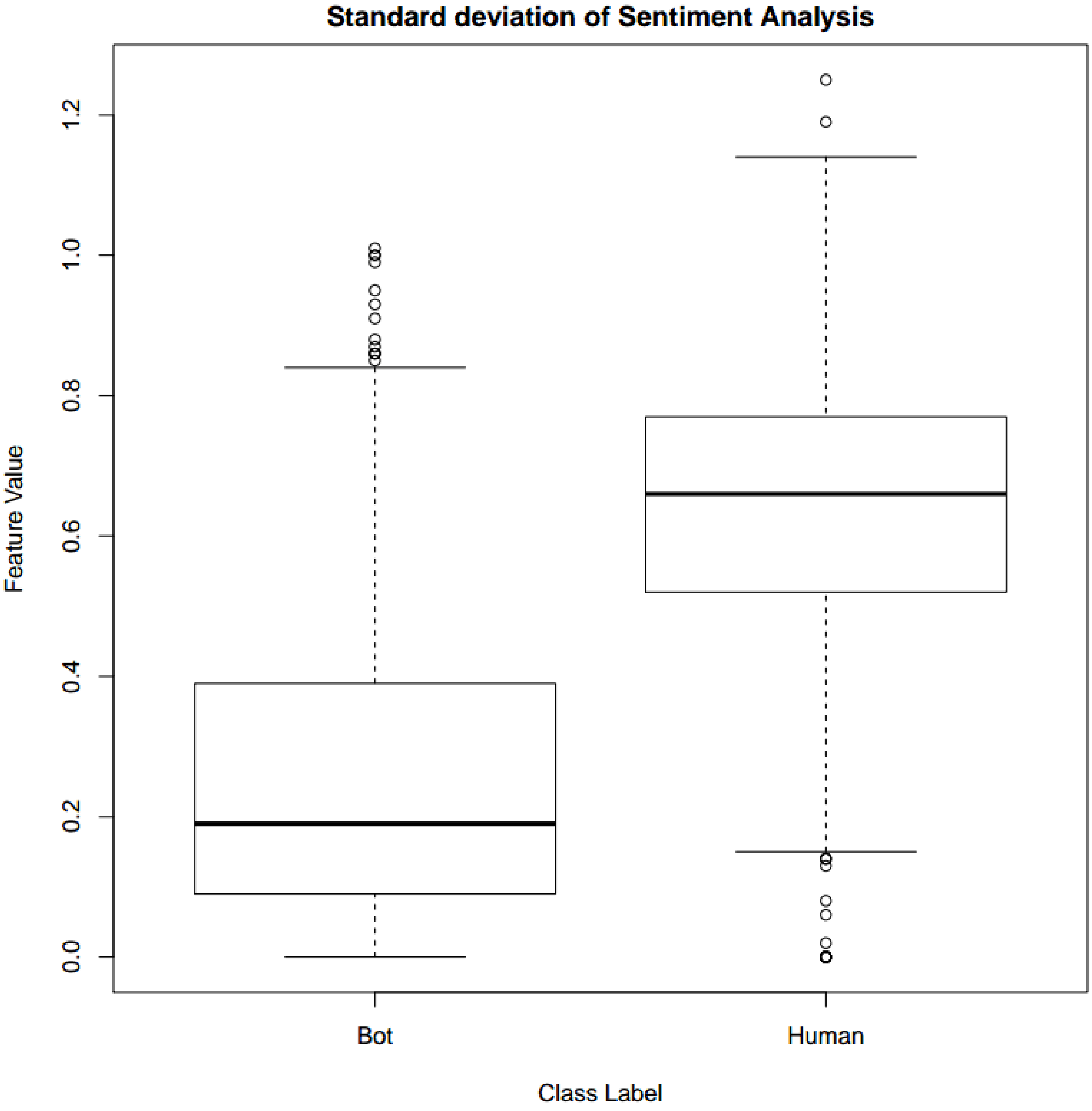
APPENDIX A

```
SELECT `screen_name`
FROM `users`
INNER JOIN `tweets` ON `tweets`.`user_id` = `users`.`user_id`
WHERE `total_friends`/`total_followers` < 0.2 AND `total_friends < 10
GROUP BY `tweets`.`user_id`
HAVING COUNT(*) >= 200
```

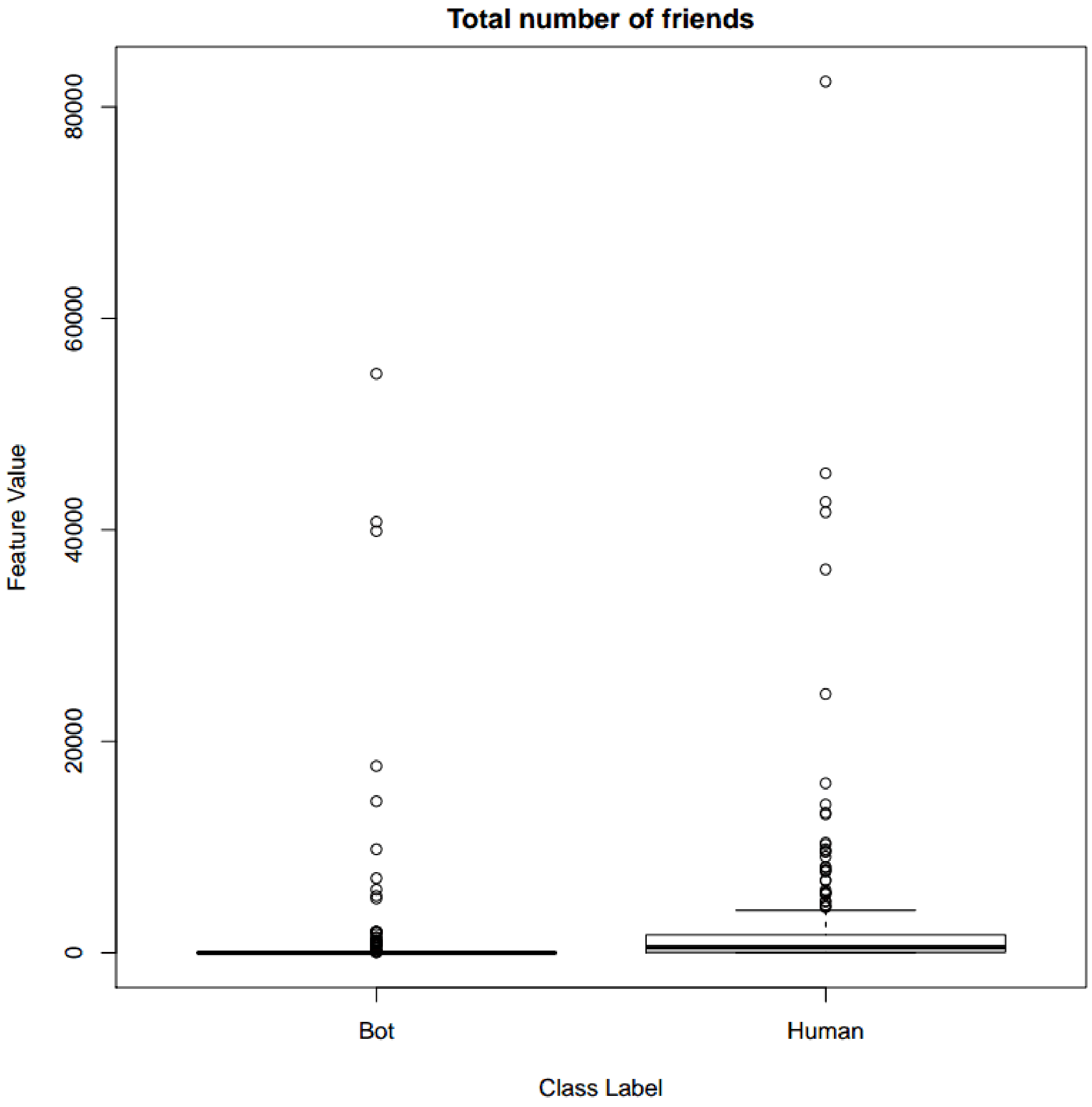
APPENDIX B.1: BOX PLOT OF THE MEAN NUMBER OF URLS



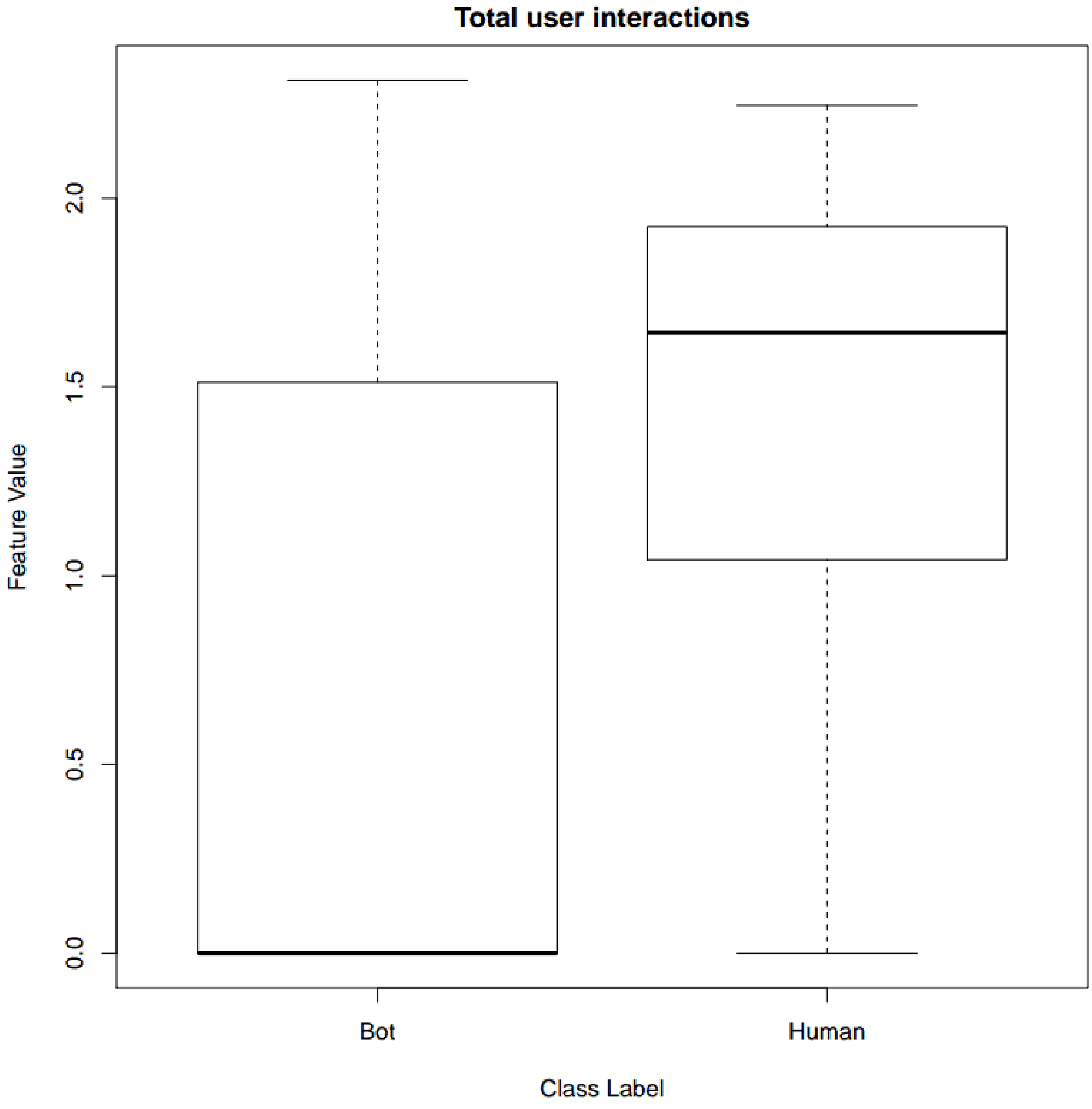
APPENDIX B.2: BOX PLOT OF THE STANDARD DEVIATION OF THE SENTIMENT



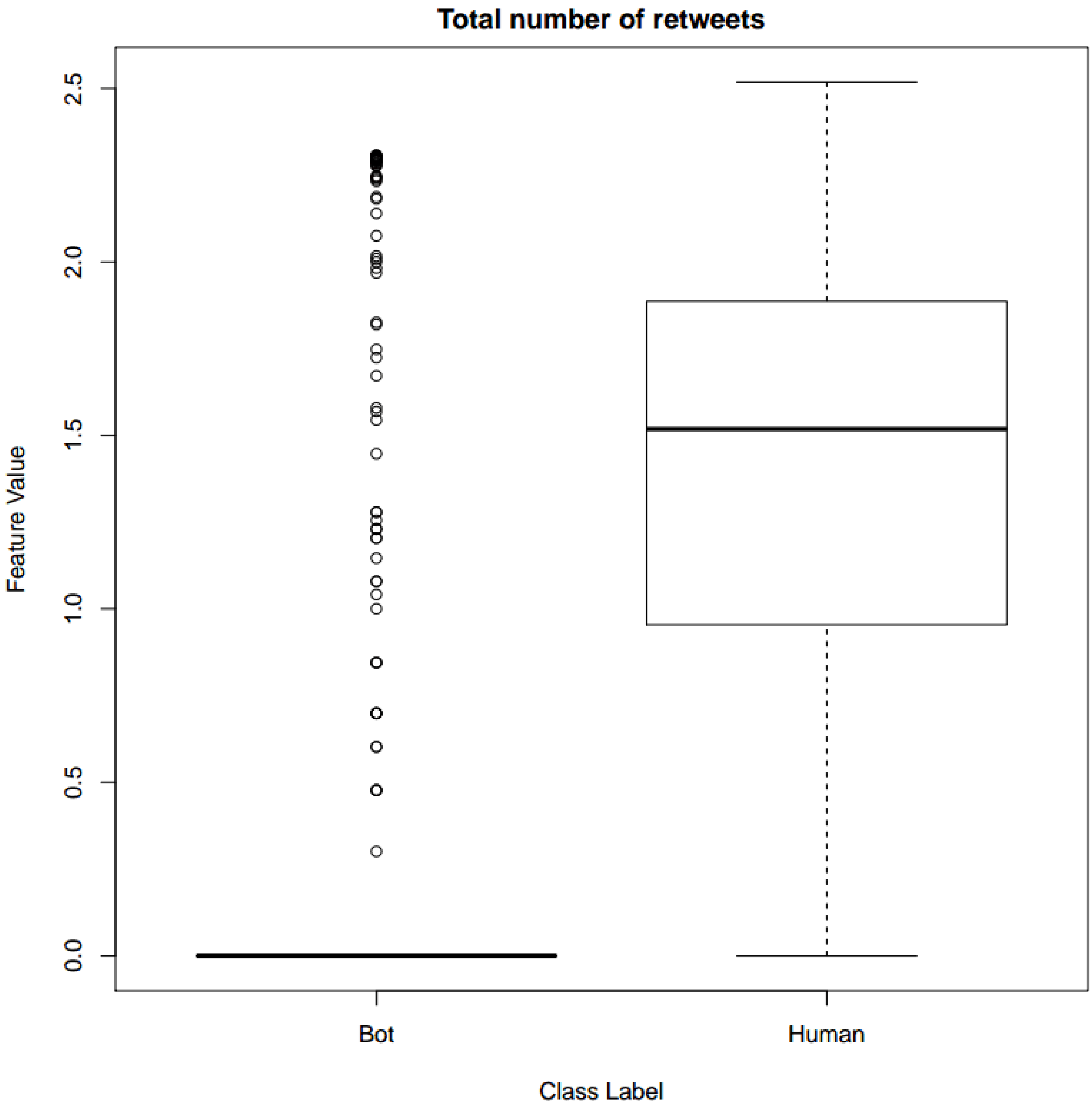
APPENDIX B.3: BOX PLOT OF THE TOTAL NUMBER OF FRIENDS



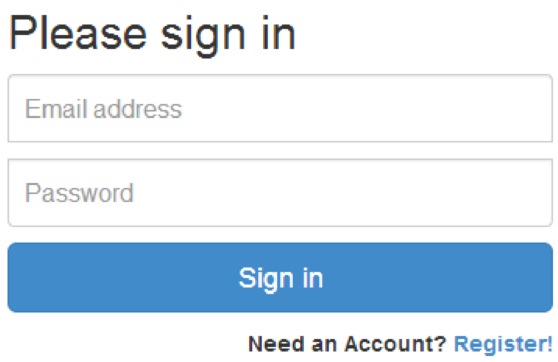
APPENDIX B.4: BOX PLOT OF THE TOTAL NUMBER OF USER INTERACTIONS



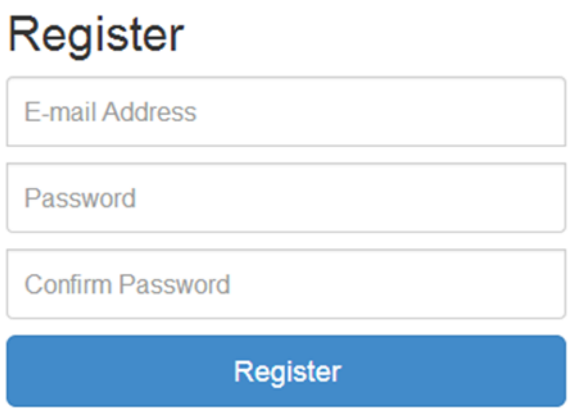
APPENDIX B.5: BOX PLOT OF THE TOTAL NUMBER OF RETWEETS



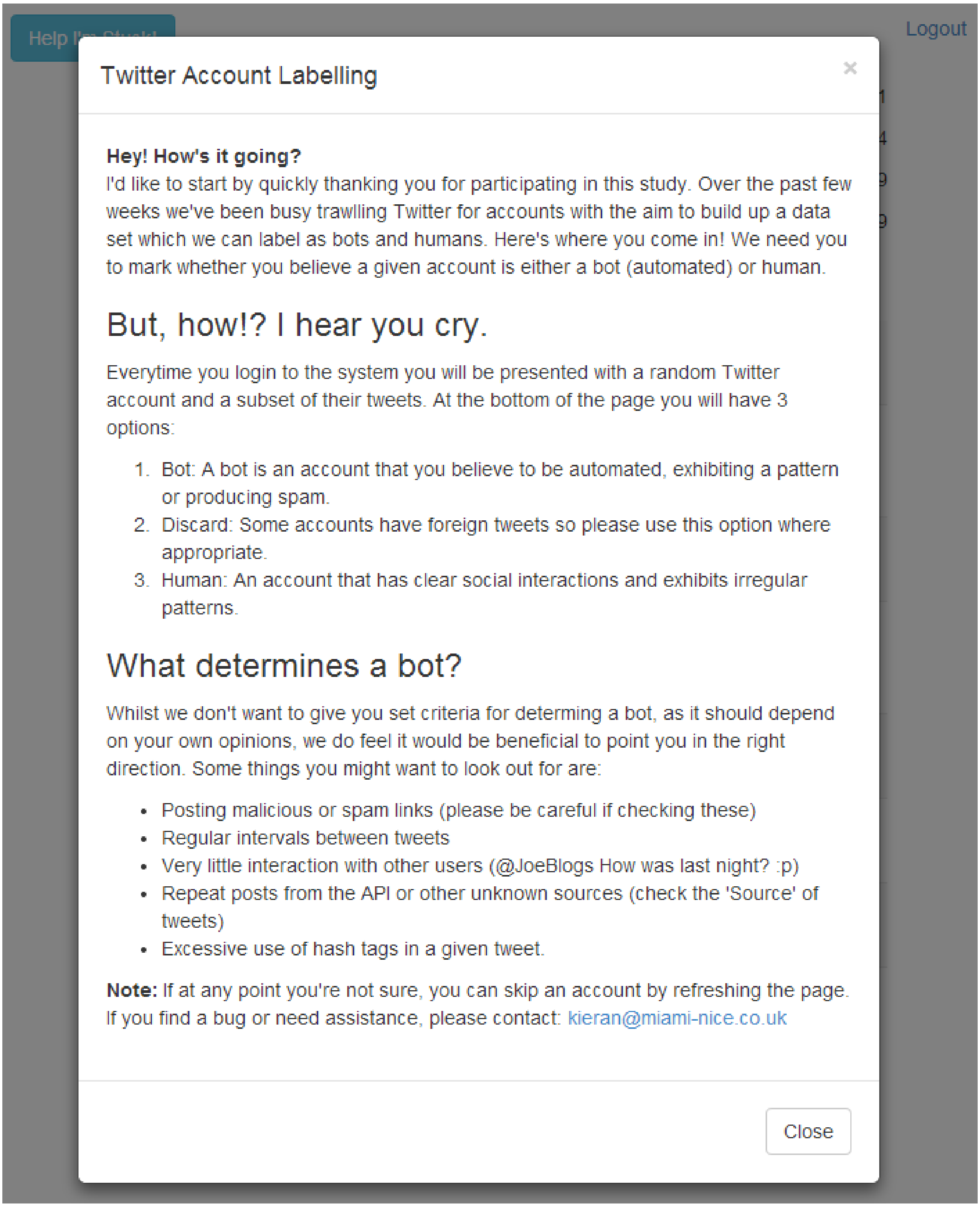
APPENDIX C: LOGIN INTERFACE



APPENDIX D: REGISTRATION INTERFACE



APPENDIX E: HELP GUIDE INTERFACE



Twitter Account Labelling

Hey! How's it going?
I'd like to start by quickly thanking you for participating in this study. Over the past few weeks we've been busy trawlling Twitter for accounts with the aim to build up a data set which we can label as bots and humans. Here's where you come in! We need you to mark whether you believe a given account is either a bot (automated) or human.

But, how!? I hear you cry.

Everytime you login to the system you will be presented with a random Twitter account and a subset of their tweets. At the bottom of the page you will have 3 options:

1. Bot: A bot is an account that you believe to be automated, exhibiting a pattern or producing spam.
2. Discard: Some accounts have foreign tweets so please use this option where appropriate.
3. Human: An account that has clear social interactions and exhibits irregular patterns.

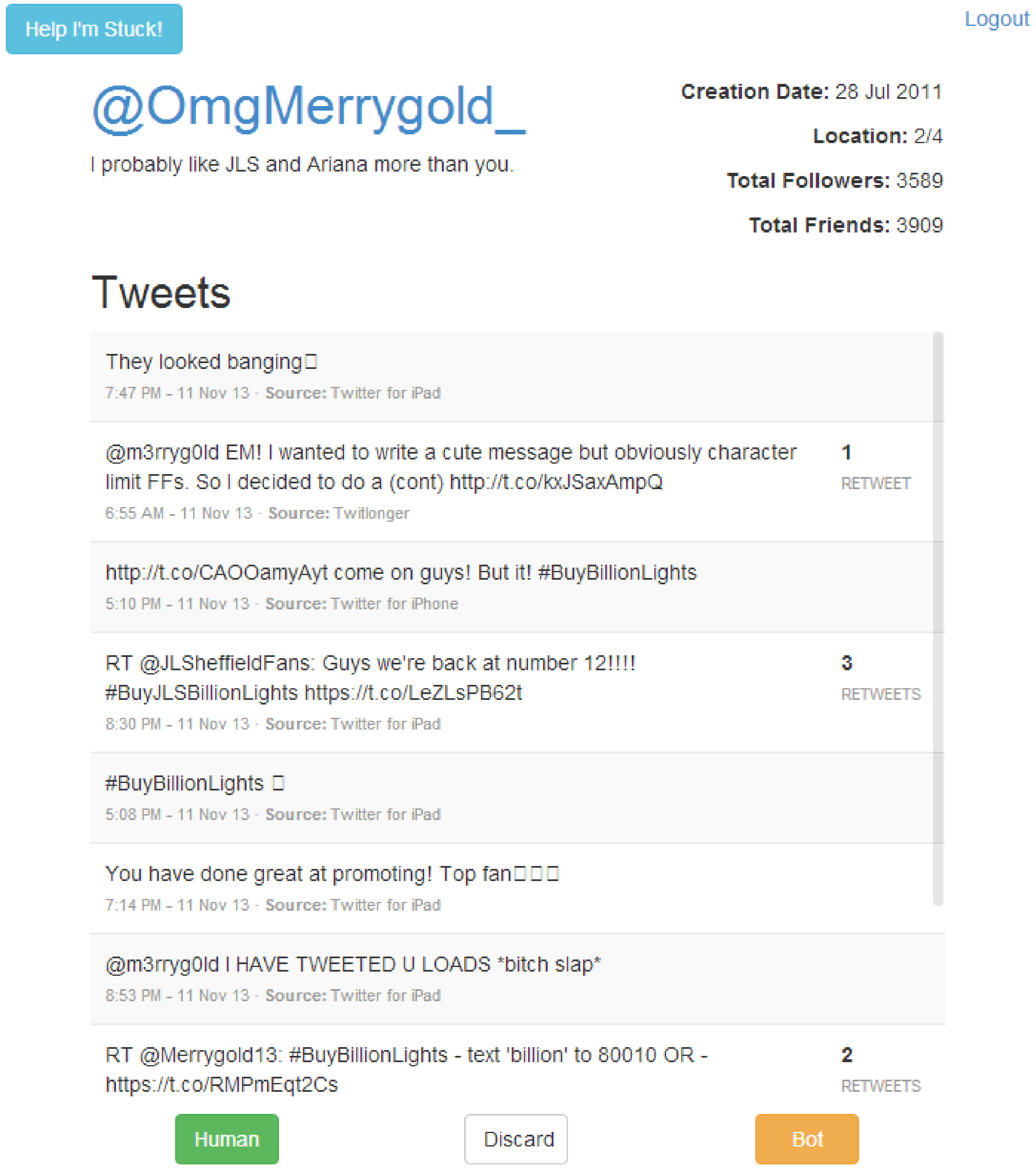
What determines a bot?

Whilst we don't want to give you set criteria for determing a bot, as it should depend on your own opinions, we do feel it would be beneficial to point you in the right direction. Some things you might want to look out for are:

- Posting malicious or spam links (please be careful if checking these)
- Regular intervals between tweets
- Very little interaction with other users (@JoeBlogs How was last night? :p)
- Repeat posts from the API or other unknown sources (check the 'Source' of tweets)
- Excessive use of hash tags in a given tweet.

Note: If at any point you're not sure, you can skip an account by refreshing the page. If you find a bug or need assistance, please contact: kieran@miami-nice.co.uk

APPENDIX F: ANNOTATION INTERFACE



@OmgMerrygold_

I probably like JLS and Ariana more than you.

Creation Date: 28 Jul 2011

Location: 2/4

Total Followers: 3589

Total Friends: 3909

Tweets

They looked banging□
7:47 PM - 11 Nov 13 · **Source:** Twitter for iPad

@m3rryg0ld EM! I wanted to write a cute message but obviously character limit FFs. So I decided to do a (cont) http://t.co/kxJSaxAmpQ
6:55 AM - 11 Nov 13 · **Source:** Twitlonger
1 RETWEET

http://t.co/CAOOamyAyt come on guys! But it! #BuyBillionLights
5:10 PM - 11 Nov 13 · **Source:** Twitter for iPhone

RT @JLSheffieldFans: Guys we're back at number 12!!!! #BuyJLSBillionLights https://t.co/LeZLsPB62t
8:30 PM - 11 Nov 13 · **Source:** Twitter for iPad
3 RETWEETS

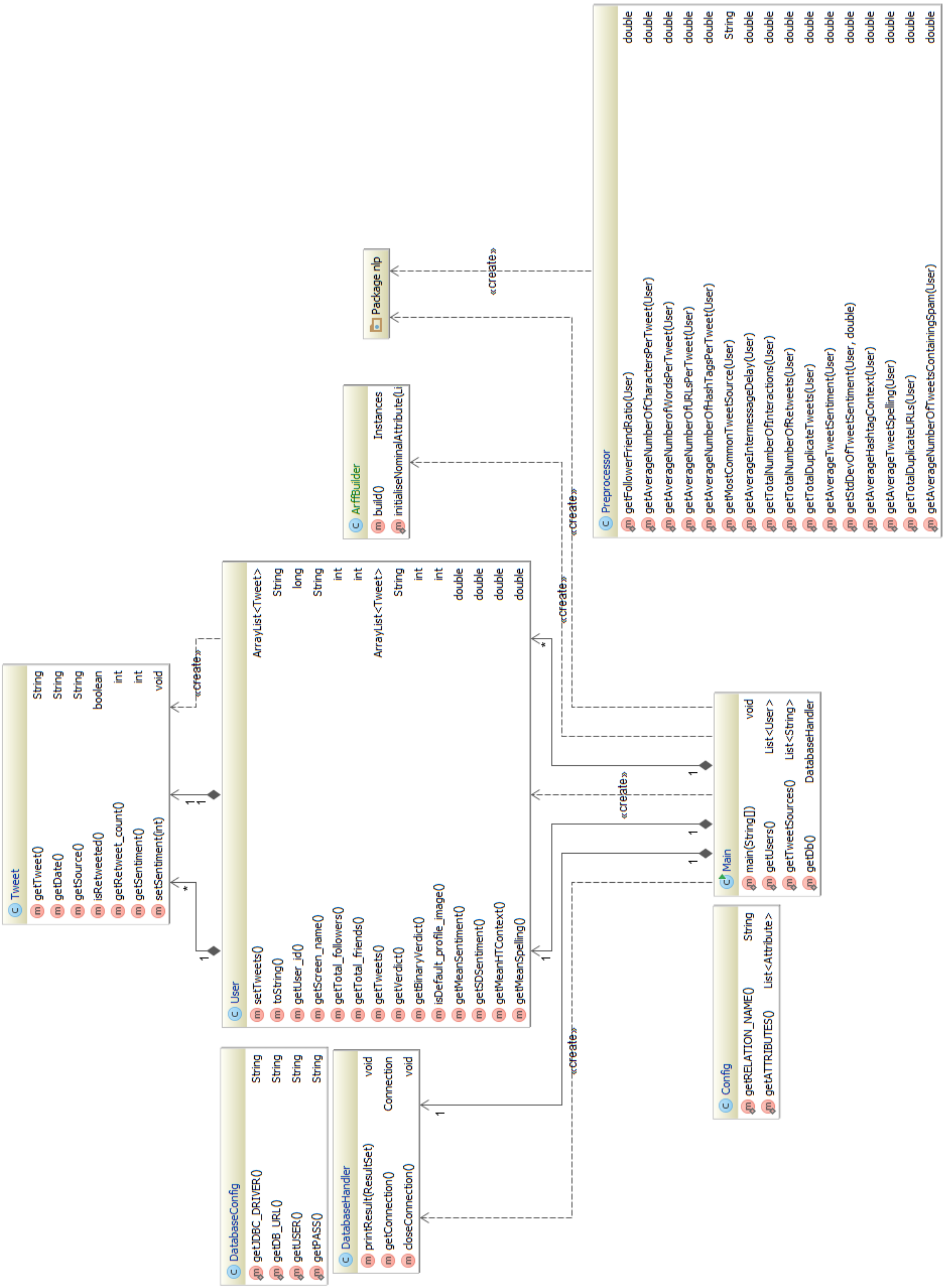
#BuyBillionLights □
5:08 PM - 11 Nov 13 · **Source:** Twitter for iPad

You have done great at promoting! Top fan□□□
7:14 PM - 11 Nov 13 · **Source:** Twitter for iPad

@m3rryg0ld I HAVE TWEETED U LOADS *bitch slap*
8:53 PM - 11 Nov 13 · **Source:** Twitter for iPad

RT @Merrygold13: #BuyBillionLights - text 'billion' to 80010 OR - https://t.co/RMPmEqt2Cs
2 RETWEETS

APPENDIX G: CLASS DIAGRAM


Package nlp
«create»
ArffBuilder
build() Instances
initialiseNominalAttribute(Li
Preprocessor
getFollowerFriendRatio(User) double
getAverageNumberOfCharactersPerTweet(User) double
getAverageNumberofWordsPerTweet(User) double
getAverageNumberOfURLsPerTweet(User) double
getAverageNumberOfHashTagsPerTweet(User) double
getMostCommonTweetSource(User) String
getAverageIntermessageDelay(User) double
getTotalNumberOfInteractions(User) double
getTotalNumberOfRetweets(User) double
getTotalDuplicateTweets(User) double
getAverageTweetSentiment(User) double
getStdDevOfTweetSentiment(User, double) double
getAverageHashtagContext(User) double
getAverageTweetSpelling(User) double
getTotalDuplicateURLs(User) double
getAverageNumberOfTweetsContainingSpam(User) double
Tweet
getTweet() String
getDate() String
getSource() String
isRetweeted() boolean
getRetweet_count() int
getSentiment() int
setSentiment(int) void
User
setTweets() ArrayList<Tweet>
toString() String
getUser_id() long
getScreen_name() String
getTotal_followers() int
getTotal_friends() int
getTweets() ArrayList<Tweet>
getVerdict() String
getBinaryVerdict() int
isDefault_profile_image() int
getMeanSentiment() double
getSDSentiment() double
getMeanHTContext() double
getMeanSpelling() double
Main
main(String[]) void
getUsers() List<User>
getTweetSources() List<String>
getDb() DatabaseHandler
Config
getRELATION_NAME() String
getATTRIBUTES() List<Attribute>
DatabaseConfig
getJDBC_DRIVER() String
getDB_URL() String
getUSER() String
getPASS() String
DatabaseHandler
printResult(ResultSet) void
getConnection() Connection
closeConnection() void
«create»
«create»
«create»
«create»
«create»
1
1
1
*
1
*
1
1
1
1
1

APPENDIX H: PROPOSAL

Social Bot Identification on Twitter

Kieran Brahney

Abstract. Social media is becoming an increasingly popular area in the public domain, with this comes bots trying to reach out to these audiences. This report presents a proposal for a project identifying between humans and robots on the social networking platform, Twitter with increased focus on bot taxonomies. The aims of this project are to produce a system that utilises existing machine learning and pattern analysis libraries to accurately differentiate these two types of user. Specifically the complete system will have the functionality to analyze a given dataset of user tweets and pull out which are robots, then perform further statistical tests in the form of outputted graphs about the dataset for the user. The project will involve two phases: gathering of the datasets and building the analysis software. Expected outcomes of the proposed system will include the ability to accurately and efficiently detect bots in a micro-blogging environment.

1 Introduction

Online social media hit the world by storm in the early 2000's with sites such as Friendster and MySpace, ever since then social media has been on the increase and only recently has it reached a truly global audience. Twitter, the network this report focuses on, was founded in June 2006 as a micro-blogging platform. As with all computer networks malicious users and spam gain prevalence as they become more popular due to the social reach one can get through these new real-time mediums.

The aim of this report is to design and implement an analysis system to accurately and efficiently determine between bots and humans on the social network. The system should allow for a user to review a given Twitter users profile using a machine learning and statistical analysis techniques to determine bot or human. Numerous types of bot exist including malicious spam, company marketing and impersonation bots. The proposed solution aims to determine type of bot and provide a risk score of the account through an easy to use yet comprehensive interface.

This report covers the background how Twitter works in comparison to other social networking mediums, the effects of bots on the social network and the implications of the resulting analysis software. Before detailing the proposed system existing research methods will be scrutinised along with previous data collection methods. The proposed system section covers the details of how the system could be developed, it's features and how the end product can be evaluated. This includes details the main areas of development and any resources required to complete the development. Finally a Gantt chart illustrating a breakdown of the tasks involved to complete the system from initialisation to completion.

2 Background

Twitter is a micro-blogging service allowing members to make, short, 140 character long updates. The service allows a modified interpretation of existing social networks using an opt-in style whereby one can be followed by ‘n’ people but one chooses who they follow (receive updates from) – those of whom there is a 1 – 1 relationship are considered friends. Other differentiating features of the service include the introduction of hash tags whereby uses can tag tweet content to a chosen category, categories used by multiple users are then considered trending in a real-time feed.

As of February 2013 Twitter was announced [8] the fastest growing social network with a 40% growth in accounts between Q2 and Q4 of 2012, with a total of 288 /485 million active accounts. With growth comes spammers, whom have various motives such as wanting to publish malicious links, automatically spread news and promotions but also hijack trending topics. Twitter has tried a number of techniques to limit spam such as the “Report as spam” feature on tweets, but regularly legitimate accounts are caught up in their actions.

Many Twitter users are unaware of the fact that a large proportion of the users they interact with on the social network are in fact bots. A study identified in [4] analyzed a selection of the leading global brands for traits similar to those of bots, which revealed 46% of whom were just that. Companies are increasingly moving to Twitter to take advantage of the fastest growing social network, thus improving marketing reach but also providing customer interaction on a much more personal level along with increased response times to customer queries.

2.1 Bot Taxonomies

A number of bot taxonomies exist on the web ranging from your traditional spam/malicious bots attempting to spread malware, company bots whose aim is to assist and automate their customer relationship management, and impersonation of high-profile figure bots each of which possess slightly different feature sets.

2.1.1 Company bots

Companies make extensive use of bots on social networks, as evidenced in [4], for help maintain their customer relationship management. Typically these bots will consist of human written message triggered in reply to the given context of a customer enquiry. They can also be used for advertising campaigns by regularly making discounts for products. Though these can be considered a good form of bot, another exists within the scope of companies which can be used to boost hash tags into trends topics in order to increase advertising campaign reach. The other being used against competitor companies to provide bad marketing through trending topics or spamming bad reports.

2.1.2 Malicious/Spam bots

Malicious bots have existed on the internet for years in the form of zombies, that spread malware. The increase of social media usage has improved the social reach for malicious bots to spread. These bots would typically spread a HTTP link to malicious software, phishing pages or counterfeit products.

2.1.3 Impersonation bots

Impersonation of high-profile users is another problem on Twitter, whereby bots are used to gather a user’s tweets then rebuild and post them appearing as that user. This can be a key problem for social engineering attacks. Twitter attempted to solve this problem a system introduced in 2009 [5] that reviewed high-profile accounts and marked them as ‘verified by twitter’ though the system is never foul proof (review process takes time.)

2.2 Related Work

Spam detection and identifying robots has been studied for a long time, with increasing focus on social media. Existing approaches to identifying bots on Twitter have focused mainly on analysing a given users trends over a period of time. In [1], Alex Hai Wang proposes two solutions using graph-based and content-based analysis to determine bots on Twitter. Graph-based analysis compares the number of friends (you both follow each other), follower ratio and the number of followers for a given account. For example you would be considered a bot should you have a small follower to following ratio. Content-based analysis makes use of the Levenshtein distance comparing the differences between tweets, though he only looks for identical tweets (distance of zero.) For example you would be considered a spam bot should you have a number of identical tweets.

Benevenuto, et al. [2] take a slightly different approach based on user trends and trending topics. Section 2 explained the emergence of trending topics and their exploitation, with this you can analyse trends of hash tag meaning in relation to tweet context. For example a typical spam tweet might be “$10.12 Viagra #will.i.am” whereby it would be put into the trending feed though completely unrelated. User trends involved the analysis of tweets across the timeline showing the number of URLs or hash tags within tweets, how often they interact with other users (through re-tweeting, replying), length of tweets, number of users replied to within a tweet and so on. Most importantly analyzing the context of a tweet in relation to the time of day as noted in [3] can be a clear indicator of whether a user is human or not. A typical human would have a pattern of how often they post throughout the week, the times of day they post at and typically a close circle of people they interact.

In [4] a slightly different approach is taken in identifying the humans and thus those left behind are the bots. This technique included features such as whether the profile contains an image, the amount of punctuation used, the context of the details in the profile description.

2.3 Implications

The importance of this report is evidenced through trending topic hijacking whereby one can quite rapidly make bad press for a competitor company, or crash financial institutions through mass panic acts. Likewise, spam detection in the e-mail world has been tackled for a years, but social media is becoming a modern form of e-mail interact with no limitations, at current, for spam. Detecting bots is the main aim of this report, how an entity acts on the returned analytical results is down to them.

3 Data Collection

To evaluate bot detection methods on Twitter an initial data set, with a clear outline of which are bots and humans, will be provided from previous research attempts into this area. Using said data experiments including but not limited to language analysis will be carried out in order to build up an accurate picture of the types of features existing bots possess in comparison to humans. The results of which can be used to determine models for the machine learning tasks of the analysis system.

Upon completion of initial analysis another data set should be gathered using other means. Data collection includes making use of fake account companies (Section 3.2) Once the data set has been assigned to the test Twitter account by the company, the Twitter API will be used to collect detailed information about user profile information, user's timeline of tweets, number of followers and following along with further collecting datasets from the accounts following the 'fake' ones assigned by the company. Thus ultimately the algorithm will span outwards, stopping at a given depth.

Tweets collected by the algorithm will be stripped of tags usually associated with Twitter such as "RT" (standing for re-tweet), "@username" (replying to another user) and "#hashtag" (tagging tweet to a context.) Hash tags and re-tweets will be individually stored in a separate array.

3.1 Twitter API

The software build through the project will be produced in Java and thus a suitable library has been found in [6] for interfacing with the Twitter API. Due to the nature of the Twitter API rate limiting requests as noticed in [7] data collection will be performed over a week long period.

3.2 Fake Follower Companies

Fake account companies have arisen in recent years that provide people with 'n' amount of a followers, of which a number can be considered bots, for a given price. A number of these companies claim With Twitter's increase in popularity, and the increase of companies setting up profiles on the social network which have begun measuring the success of their reach based on the number of followers they have (who ultimately see's their posts.)

4 The Proposed System

The aim of the proposed system is to design and implement a software package that can accurately and efficiently determine between bot and human on Twitter. The system will be developed in Java making use of existing machine learning libraries such as [9] and Twitter API libraries such as [10]. The main components of this system will entail initial user input, background interaction with the Twitter API (data gathering), analytics using machine learning and a final resulting interface for Desktop machines (though being Java would be available on other platforms.)

4.1 Interface

The interface for the software needs to be easy to use yet comprehensive enough to gain a good insight into the characteristics of a given users Twitter account. Initially upon opening the system the user would be asked to provide a Twitter username to analyse. The system will begin gathering data about the user using the Twitter API and compare it against the machine learning model in which has been developed during initial data-set experiments. The analysis should be a short, efficient process to avoid making the user wait too long. Upon completion the system will provide a probabilistic score on the likelihood of the user being a bot using Bayes' theorem. The system will also output graphs and other useful information from the resulting gathered dataset for the user to manually decide upon whether they believe tweets to originate from a bot or human.

The system should provide the ability to dig deeper into a users account by providing a table of users associated with that of the provided. Depending on the efficiency of developed algorithms this could include a risk score, calculated via Bayes', for each user. Further abilities should be provided through sorting of the associated user data set.

4.2 Features

The proposed system will use features based on the research described in section 2.2 (related work), repeating some of the tests from this research and then using machine learning techniques and statistical analysis to improve upon this analysis and find additional taxonomies that identify bots.

As mentioned in section 3, data collection, analysing of an initial data set is important to gain a clear understanding of the features sets that firstly bots in general possess, but also the features that differ between those of different type. Classification models that should be useful in determining a given bots taxonomy include firstly Logistic regression, Naive Bayes and Support vector machines before finally using decision tree classifiers to analyse the observations made by earlier classifications.

The system should make use of a two-stage framework. This will firstly involve identifying bots, and secondly differentiating between the different types of bots as noted in section 2.1. Initially this will involve a binary classification model of separating the data set between bots and humans. Once separated bots will be further analysed and passed through cluster analysis algorithms to determine the bots taxonomy. Should a given taxonomy not exist within the system, it shall classify it based on it's context using natural language processing techniques.

In partnership with the binary classification model when determining between bots and humans, but also different types of bots, statistical tests shall be carried out to better determine an outcome. The Mann Whitney and 2-sample T test can be used to compare two samples and see the significance to which they differ. For example, as mentioned in section 3 comparing expected features of a given bot taxonomy against the actual data set.

Existing techniques will be analysed in depth via research of the theorems and methodologies involved, such would involve manual investigations and calculations on existing data sets to draw the accuracy and efficiency from these methods. Should results prove to be conclusive the method will be used. As the project progresses examination of further, new, ideas will involve comparing them to existing techniques but also manually reviewing them against real Twitter data before further proving said idea against its associated theorem.

Ideas that have yet to be considered in prior research include using Natural Language Processing to determine the context of the tweet in relation to the category given by the hash tag, thus things completely unrelated would increase the potential bot score. Similarly, online analytics of websites has dramatically improved in determining bad offenders on the web. Thus, it is proposed gathering the risk score of a given URL as a determining factor to the risk of a given tweet.

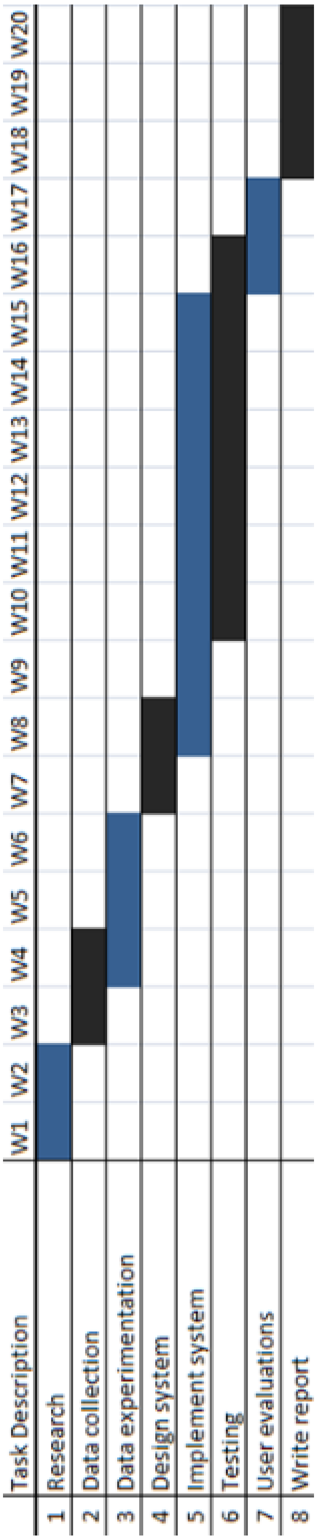
5 Evaluation

Measuring the success of identifying a bot on Twitter is the most important of the system life cycle. In order to evaluate the overall success of the system two evaluations can be made, these being performance-based and usability-based.

Performance-based evaluation will involve measuring how good the system is at first detecting a bot but secondly classifying the bot into it's given taxonomy. For this classification measures will be used being precision and recall (system produces a things it believe to be bots and of a given taxonomy, these values can be then measured.) Such techniques will then be applied to the F1 score, calculating the test accuracy and used to refine the machine learning model.

Usability-based evaluation will involve testing how the useful the system for the end-user. This should involve a separate, online, system that allows a given test user to sign into their Twitter account and build a list of their tweets (from their timeline) in tabular form. The backend Java based system will then analyse the given data set and present back to the web application what it believes are generated by bots, along with their associated taxonomy. The user can then go through the table and mark which are correct and which are not. After which point the user would submit the results of the evaluation and be presented with a questionnaire to outline their opinion of the positives and negatives of the analysis system. Such would provide both quantitative data in form of the tabular corrected result set and qualitative results from the user questionnaire after the evaluation process.

Gantt Chart



	Task Description	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17	W18	W19	W20
1	Research	■	■																		
2	Data collection			■	■																
3	Data experimentation				■	■	■														
4	Design system							■	■												
5	Implement system								■	■	■	■	■	■	■	■					
6	Testing										■	■	■	■	■	■	■				
7	User evaluations																■	■			
8	Write report																		■	■	■

References

1. Detecting Spam Bots in Online Social. Networking Sites: A Machine Learning Approach

   http://personal.psu.edu/students/h/x/hxw164/files/DBSec2010_Wang.pdf

2. Detecting Spammers on Twitter

   http://ceas.cc/2010/papers/Paper%2021.pdf

3. Twitter Study, August 2009

   http://www.pearanalytics.com/wp-content/uploads/2012/12/Twitter-Study-August-2009.pdf

4. Nearly Half of all Twitter Followers are Bots

   http://technorati.com/social-media/article/nearly-half-of-all-twitter-followers/

5. FAQs about verified accounts

   https://support.twitter.com/groups/31-twitter-basics/topics/111-features/articles/119135-about-verified-accounts

6. Twitter4J

   http://twitter4j.org/en/index.html

7. Twitter API Rating limit numbers

   http://twitter4j.org/en/api-support.html

8. Twitter The Fastest Growing Social Platform

   https://www.globalwebindex.net/twitter-the-fastest-growing-social-platform-infographic/

9. Twitter4J

   http://twitter4j.org/en/index.html

10. Weka – Data mining

    http://www.cs.waikato.ac.nz/ml/weka/