CATEGORIZING BLOG SPAM

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Brandon Bevans

June 2016

COMMITTEE MEMBERSHIP

TITLE:	Categorizing Blog Spam
AUTHOR:	Brandon Bevans
DATE SUBMITTED:	June 2016
COMMITTEE CHAIR:	Foaad Khosmood, Ph.D. Assistant Professor of Computer Science
COMMITTEE MEMBER:	Alex Dekhtyar, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Bruce DeBruhl, Ph.D. Assistant Professor of Computer Science

ABSTRACT

Categorizing Blog Spam

Brandon Bevans

The internet has matured into the focal point of our era. Its ecosystem is vast, complex, and in many regards unaccounted for. One of the most prevalent aspects of the internet is spam. Similar to the rest of the internet, spam has evolved from simply meaning 'unwanted emails' to a blanket term that encompasses any unsolicited or illegitimate content that appears in the wide range of media that exists on the internet.

Many forms of spam permeate the internet, and spam architects continue to develop tools and methods to avoid detection. On the other side, cyber security engineers continue to develop more sophisticated detection tools to curb the harmful effects that come with spam. This virtual arms race has no end in sight. Most efforts thus far have been toward accurately detecting spam from ham, and rightfully so since initial detection is essential. However, research is lacking in understanding the current ecosystem of spam, spam campaigns, and the behavior of the botnets that drive the majority of spam traffic.

This thesis focuses on characterizing spam, particularly the spam that appears in forums, where the spam is delivered by bots posing as legitimate users. Forum spam is used primarily to push advertisements or to boost other websites' perceived popularity by including HTTP links in the content of the post. We conduct an experiment to collect a sample of the blog posts and network activity of the spambots that exist in the internet. We then present a corpora available to conduct analysis on and proceed with our own analysis. We cluster associated groups of users and IP addresses into entities, which we accept as a model of the underlying botnets that interact with our honeypots. We use Natural Language Processing (NLP) and Machine Learning (ML) to determine that creating semantic-based models of botnets are sufficient for distinguishing them from one another. We also find that the syntactic structure of posts has little variation from botnet to botnet. Finally we confirm that to a large degree botnet behavior and content hold across different domains.

ACKNOWLEDGMENTS

Thank you to my advisor Dr. Foaad Khosmood for the constant guidance and the initial idea. This thesis introduced me to many new concepts in Computer Science that I have grown a passion for, and for that I am very grateful.

Thank you to my family and friends for their unconditional love, support, and guidance.

For George.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER1

INTRODUCTION

1.1 What Is Spam

Spam has grown from simply meaning 'email junk' to a general term used to refer to any unsolicited commercial communications. As the internet has evolved, spam has become much more advanced in both its sophistication and breadth. There are many reasons why spam is detrimental to the well being of the internet and its users. For one, it consumes valuable bandwidth resources. In 2015, over half of all emails sent were spam [24]. This costs precious processing time for email servers. It also costs person hours to keep spam filters up to date, and for consumers to manually process spam that makes it through the filters. Additionally, when spam isn't detected by spam filters or the victim, it can often lead to the download of malware and the loss of sensitive information. Spam is even used to deliver harmful malware such as ransomware, which holds the victim's data captive until a fee is paid. Additionally, spambots, the agents responsible for the delivery of spam, have much more capability now. This includes automated forum registration, confirming email verifications, and even passing tests that are specially designed to identify them. As a consequence, detecting spam has become both more difficult and more important.

1.2 Blog Spam

Currently, a common use for spambots is to pose as a forum attendant, who contributes new posts or responds to the posts of other users. This subcategory of spam is referred to as 'forum spam' or 'blog spam'. A spambot has two objectives when posting blog spam. The first is advertisement. Because blog spam is posted in actively browsed websites, other users will likely observe it, so the spam constitutes a channel for which the spambot can advertise to users. The second objective of blog spam is to boost the perceived popularity of a target website to search engines. The way the blog spam accomplishes this is by including link spam in it's content. Link spam is any HTTP link that exists within the content of the spam. The link spam is observed by a search engine's crawlers, and then the link's target receives a boost in perceived popularity. This leads to an overrepresentation of how popular the target website is, which degrades the service that the search engine provides. The goal of this thesis is to gain a deeper understanding blog spam and where it comes from.

1.3 Our Contribution

In the past, the vast majority of efforts regarding spam have been toward prevention. Email spam filters are generally effective, and there are reasonably effective safeguards available to tell between a human and a robot when performing critical actions on a website, such as posting content or registering an account. Very little research has been done to characterize the spam on a macro level. In this thesis, we set up an experiment to collect a sample of the forum spam present in the wild. To do this, we set up three honeypots, or fake websites, that appear legitimate to spambots. Each one is homogenous with the exception of the theme that the first contains, which was purposely put there by us to see if it influences the content that the spambots post. The observation period of the experiment is 42 days. We observe the data

left by the spambots. This includes the network activity, the actual content of the spam posts, and the user information for all of the users that were registered by the spambots.

After we conduct the experiment, our research commences. Our research results in the following contributions:

- **Corpus:** From the data that we gathered in the experiment, we form a corpus from which to conduct the rest of our analysis from. The corpus consists of three tables for each honeypot. The corpus is made publicly available for other researchers to conduct analysis on. The tables are described below:

  User Table: The User table contains all of the log in information of the users that the spam bots created. This includes the username, the number of times they logged in, and all of the IP addresses and respective geolocations that they logged in from.

  Access Table: The Access table describes all of the network requests logged from the honeypots. It contains raw data such as the request path, the source IP address, and the request type, and also some lightly processed data such as the desired 'action' of the request.

  Content Table: The content table houses all of the blog posts and comment posts that were made during our experiment. Contained in each entry is the raw content of the post, the type of post, the time of the post publication, the ID of the user who posted, and some other useful metrics such as the number of hits it received, a list of links that were in the post, and a taxonomy of the posted content.

- **Entities:** We model the botnets that interact with our honeypots. We do this in order to understand the underlying mechanisms that generate blog spam. For

each honeypot, we combine users who are directly or indirectly connected to each other into collections called 'entities'. A connection in this case is defined as any user who shares an IP address with another user. The result is a model of the botnets that interact with our website, which provide grounds for the classification analysis that we perform next.

- **Classification:** Using the entities outlined above, we determine the most distinct features of each entity through the use of Natural Language Processing (NLP) and Machine Learning (ML). We derive a number of feature sets that express a particular attribute of the content posts, such as the Part Of Speech (PoS), which represents the syntactical structure of a post, or the Bag Of Words (BoW), which gives an idea of the semantic content of a post. We find that among the attributes we consider, semantic modeling is the most effective way of distinguishing one entity from another, both by using the actual words present in the content of the spam, or more efficiently, a derived taxonomy of the content.

- **Behavior:** We then combine all of the honeypot-specific entities into 'meta entities', a new class of entities that are distinct across all honeypots. We use the same formation process as before, only this time with the entities instead of each honeypot instead of the users. We find a number of the active entities are present in each honeypot, and then perform a similar classification process on the meta entities. We find that some of the semantic models, namely BoW, have nearly the same classification accuracy in a domain-agnostic setting as it does in a domain specific one. We take it a step further by dividing the feature sets into domains, and then introducing a test set whose domain is completely unknown to the classifier. The BoW classifier is slightly less accurate, yet still adequate in determining which meta entity the posts come from, which suggests that our classification system could be installed into a fresh website

and still gain accurate classification information. Additionally, in some meta entities there were multiple entities from the same domain. This means the IP addresses gained from the other honeypots effectively linked two entities within the same honeypot that our model considered separate before. The combination of these two findings suggest that a central database with many distributed data collection points could potentially model the entire internet's botnet ecosystem.

The rest of this paper is organized as follows:

Background: This section provides an overview of the technologies and ideologies we use to design our experiment and analyze our findings. It also covers related works that are relevant to our cause. This includes a review of spam in general, as well as honeypots, natural language processing, and machine learning.

Tools & Technologies: This section describes the various 3rd party packages that we use to aid us in our findings. We use various programs, libraries, frameworks, and APIs to help with our honeypot setup, NLP, and machine learning analysis.

Experimental Design: This section goes into detail about how our experiment is set up. It describes what Content Management System (CMS) and server configuration we use, the time span we run the experiment for, and the data set we collect. It also explains the reasoning behind our design decisions and the initial hypothesis in mind when forming our experiment.

Results: This section describes: the corpus we formed from the experiment, the methods used for analyzing our data, and all of the findings that result from our analysis.

Conclusion: This section recaps the experimental process, the analysis techniques we used to analyze the data we collected, and the results that find.

Future Work: This section suggests future work that could be done based on the findings of this experiment.

CHAPTER2

BACKGROUND

2.1 Spam

The definition of spam continues to evolve. Once simply meaning 'email that is not wanted', a more broader definition is now required, such as 'any unsolicited commercial communication'. As the internet becomes increasingly complex, so does spam. There are now many variations of spam that attackers use, including 'blog' spam. Blog spam, also known as 'forum spam' or 'splogs', is disingenuous content posted to an otherwise legitimate forum, presumably made by a program rather than a human user. Blog spam has become an increasingly popular avenue for spammers [25]. Blog spam is used for a variety of purposes. Many sources agree that the two main motives of generating splogs are to create a source of profitable advertising and to create a link farm that unjustifiably increases the PageRank of affiliated sites [13][25]. A site's PageRank is a score assigned by Google in order to determine the importance of a website. When a site's PageRank increases illegitimately it causes search engines to rank the page as more important than their legitimate PageRank warrants. In addition to blog spam, comment spam is a popular channel for spammers. In 2007, the spam classification service Akismet classified 95% of the submitted comments it received as spam [25]. Although spam is a big problem, research shows that a

variety of detection methods are effective in detecting spam. This includes focusing on certain metrics of spam that distinguish it from legitimate, normal content, often times referred to as 'ham', . One study [9] found that training classifiers with a specially added malicious feature set significantly improved the classifiers ability to detect spam, even when the spam only accounted for 16.5% of the dataset.

Researchers have gone through great lengths to find out how the underlying mechanisms of spam work. In one study [22], the researchers fully evaluated how one of the most popular spam automation program works. The program, called XRumer, offers a suite of tools that provide a huge amount of utility to the botmaster, or the party responsible for deploying a spam campaign. XRumer includes means for target collection, automated forum registration and posting, and detection avoidance. It also has the utility to use proxies and integrate other popular spam tools to gain more functionality. Among these other tools, Hrefer stands out as a crucial tool for the target collection process. To find target forums, XRumer allows the user to scrape Google, Yahoo, and a number of other search services. Hrefer is the tool that allows the user to do this, and it comes free with the purchase of XRumer. Hrefer uses the standard APIs that all of these services offer to get a comprehensive list for the spambot to target. Hrefer does this in a sophisticated way, accessing the API in a way that makes its detection difficult. Hrefer also makes searching easy for the user; based off of a given keyword, it infers more keywords through the malevolent use of Google AdWords Keyword Tool [10]. For the actual generation of spam, XRumer allows the use of macros and macro variation, essentially allowing the user to produce a lightweight grammar for post variation. Finally, XRumer incorporates the tool Webalizer to incorporate what's known as 'Refspam', which allows the spam to boost a target URL's PageRank through malicious HTTP header manipulation, even if the spam itself is detected. There are other tools that mimic what XRumer is used for, but it appears that XRumer is the most popular and effective. The authors also pro-

vided a graph that describes the basic process when determining if spam is legitimate or not, as can be seen in Figure 2.1.

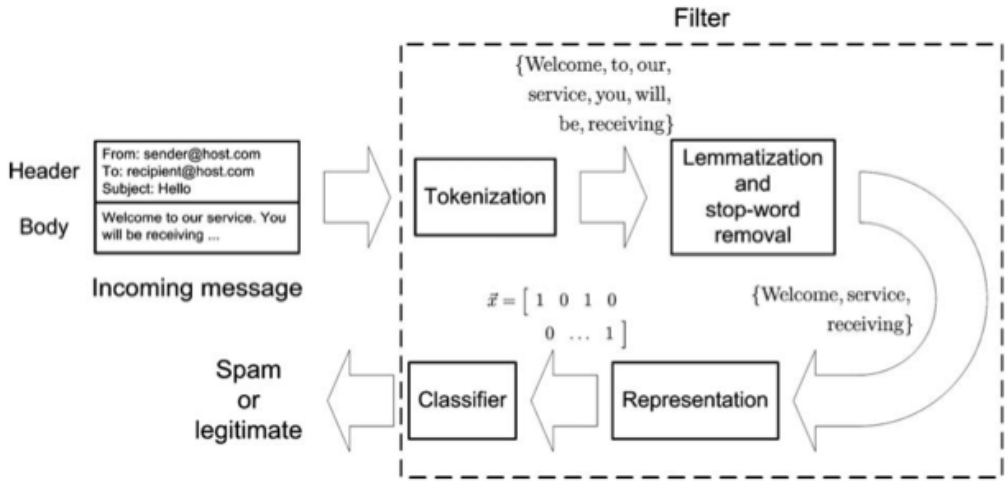


Figure 2.1: The Main Steps Involved in a Spam Filter [11].

Other research has focused more on how to detect and reverse the specific effect that spam has on search engine ratings. In [2], a program called SpamRank is introduced that successfully detects and mitigates the effect of PageRank boosting. Their success depended on two key properties of the internet that allowed them to automatically detect link spam abuse with success. The two properties they find are as follows:

- Portions of the web are self-similar. For any sufficiently large set of honest supporter pages, a representative graph of it will look similar to any other sufficiently large set, even if the sets in question are completely disjoint. Put in a more specific way, the PageRank of distribution of the pages that link and promote a given page will naturally follow a power law distribution, and this trend follows for any given page of the web. Contrarily, if a page receives its PageRank from only very low ranked pages, and a high volume of them, this is regarded as suspicious because of the flat distribution; the supporter doesn't fit into power law.

- Link spammers have a limited budget; when boosting the PageRank of a target

page, 'unimportant' structures are not replicated (like the power law distribution).

This work was focused on how to improve PageRank algorithms to detect and penalize a page whose rank has been inflated illegitimately. It would be applied to search engine algorithms and other website scraping applications. Our work is peripheral to this. Rather than detect if a page has been spam-boosted, we analyze and characterize the very spam campaigns that are responsible for page boosting.

Another point of interest within the topic of spam is the underground ecosystem built around it. One group of researchers [23] was able to infiltrate some inner circles of botnet operators and gain knowledge about the ecosystem of spam campaigns. Through undisclosed means they were able to gain enough trust to be allowed access to protected forums and privileged information. From this, they were able to see what was valuable to spammers, such as active email lists. As a whole, their infiltration led to three major contributions, listed below:

- Deep analysis of the Cutwail spam operation (one of the biggest botnet programs known on the internet), based on information directly collected from the botmaster' s hosts. This includes a manual written for botmasters on how to manage the botnet.

- The botmaster's role in delivering and managing a spam campaign, including the software architecture used and its role in the economy.

- The analysis of the 'Spamdot.biz' forum and the transactions of the spammers to gain an understanding of the economics of spam campaigns.

Aside from their research contributions, they also gained access to 16 servers that belonged to the botnet. In addition, through notification of the respective ISPs, over 20 malicious servers were identified and shut down thanks to the efforts of the

researchers. Our work is similar to theirs in that we are trying to understand the spam ecosystem. Their research was focused on the economy of spam campaigns, where we are more focused on the distribution and classification of spam on the internet as a whole. Additionally, we take a much less intrusive approach, analyzing only what the spam mechanisms forcibly make available to us, and without any active intrusion on our part.

The active landscape of spam is quickly evolving. As new technologies emerge to allow us to interact and communicate with each other in different ways, spam will continue to follow suit. According to the most recent Internet Security Threat Report by Symantec [24], the rate of spam is actually going down, however the spam is becoming more malicious. Instead of advertisements and promotions, spam is heading in a darker direction, being more likely to contain malware than in previous years.

2.2 Honeypots

A honeypot is a decoy system whose purpose is to lure in fraudulent cyber activity. They're used by researchers to study the techniques and heuristics that adversaries use to attack a system. They're deployed in a controlled, isolated environment, and they don't present any useful data in the case that the honeypot is compromised. To an attacker, a properly deployed honeypot is indistinguishable from a genuine system. The honeypot may have additional logging or surveillance attached to it to gain more fine-grained information about the attackers methods. Honeypots are used for a number of reasons, from gathering clusters of malicious IP addresses for blacklists, to detecting and analyzing new malware. There are two broad categories of honeypots, high-interaction and low-interaction honeypots. High-interaction honeypots allow the hacker to interact with the system as if it were any regular system. The advantage of high-interaction honeypots is that it more accurately portrays a real environment, and

therefore has the potential to gain the most information on the hacker's techniques and activities. There are minimal restrictions on what the attacker can do once they compromise the system. On the other hand, low-interaction honeypots only allow for a limited subset of functionality that they would expect. Usually the honeypot server will only support the services needed for the honeypot operators to identify that the exploit is being attempted [18]. Honeypots have a number of advantages over other security measures. This includes fewer false positives since no legitimate traffic will go through the honeypot [18]. This is crucial to our analysis since we can assume that very little to none of the traffic that come through our honeypots is legitimate traffic, and therefore speculate with the assumption that all of our data is valid.

Although we didn't require the use of an explicit honeypot framework, they could be classified as mid-interaction. In [20], a virtual honeypot daemon called Honeyd is introduced. Honeyd allows for the easy deployment of a virtual honeypot that simulates computer systems on the network level. It has the ability to simulate complex network topologies, which lets it fool network-fingerprinting tools such as Nmap. This is crucial so that attackers can't distinguish between a honeypot and a legitimate system.

In [17], a honeyclient tool, PhoneyC, is introduced. Much like a honeypot allows insight into server-side attacks, a honeyclient is a tool that provides insight into client-side attacks. PhoneyC is designed to mimic the behavior of a user-driven application like a web browser, and then be exploited by malicious content. From there it can pinpoint attack vectors and use dynamic analysis to remove obfuscation from malicious pages.

2.3 Natural Language Processing

The field of Natural Language Processing (NLP) has the goal of converting human language into a formal, well defined representation that is easily read and processed by computers [6]. Although remarkable progress has been made, it is a huge undertaking; the uncertainty of language and the discrete nature of computers do not mix well. Language in its natural format is full of exceptions to rules, interpretation, ambiguity, and a myriad of other headaches that computers must overcome in order to understand. The process of taking loosely formed language structures and extracting discrete, concrete meaning structures from it is inherently difficult. There have, however, been great strides to aid the process of digesting language. In [4], many NLP fundamentals are outlined, such as part-of-speech tagging and other methods to programmatically extract the meaning of linguistic structures.

There are also ways of determining how effective a given feature set of a language is. In [3], the authors focus on the maximum entropy principle. By following the maximum entropy principle, it allows one to select a model within a set of features that nets the greatest entropy. They also discuss algorithms for constructing a maximum entropy model. Outside of [4], there has been effort to create NLP frameworks to speed up the process of converting human language into computer-workable data structures. In [6], the authors outline a general task list that falls under NLP, listed below:

- Part-Of-Speech Tagging (POS): This step aims at tagging each individual word with a tag that represents its syntactic role within the sentence, such a noun or a verb.

- Chunking: A step up from POS, chunking is the process of labeling groups of associated words, such as noun phrases or verb phrases.

- Named Entity Recognition (NER): This step labels proper entities within a

sentence into categories such as 'person' or 'location'.

- Semantic Role Labeling (SRL): This step aims at giving semantic roles to syntactic constituents within a sentence. This step aims to disambiguate a word that could have multiple parts of speech.

- Language Models: A model is then built to estimate the probability of what the next word will be within a sequence.

- Semantically Related Words: This is the process of predicting whether two words are semantically related.

2.4 Machine Learning

The efforts of proper NLP often result in a structured set of data for which to perform machine learning (ML) algorithms on. The objective of machine learning is to learn from and make predictions on data. The most widely used form of machine learning is classification. A classifier is a system that inputs a vector of feature values and outputs a single value, the class [8]. In the context of textual data, NLP is used to form these feature values, which can then be fed as input into a machine learning system. There are multiple components to a machine learning system, such as the learner. The learner takes as input a set of training examples in which the expected answer is provided, and outputs a classifier.

We are not the first to apply ML algorithms in a spam context. In [11], the current ML approach to spam filtering is reviewed. A few common ML algorithms are explained, namely Naive Bayes classification, artificial neural networks, and logistic regression. This paper discusses machine learning in the context of spam filtering, which is where the vast majority classification efforts regarding spam have been focused. This is a peripheral topic to our research; in our context we already know that

the data we're using is spam. We're aiming for a more distinct classification. The conclusion that although progress has undoubtedly been made, there is still work to be done, particularly in the context of keeping filters updated in a cheap and accurate fashion.

When [13] was published, there were no known machine learning approaches to detecting blog spam. The techniques used at the time included much less effective techniques such as URL patterns, URL and IP blacklists, and manual checking. They introduce the first machine learning system to classify blogs. The advantages are a substantial boost in blog spam detection, and the ability to easily be retrained as the blog spam changes. Their results are a good sign for us; they indicate that ML processing is effective with blog content.

In [21], automated text classification is a major research area within information systems. The author attributes its rise to such importance to many factors, namely:

- Given the rise of digital documents, the domains of application for text classification are numerous and important, and are bound to increase in both number and importance.

- Many times, the sheer number of the documents to be classified and short response time required make the manual alternative implausible.

- It has reached effectiveness levels comparable to those of trained professionals. The rate of automated text classification effectiveness of is growing, and though they probably will not reach 100%, they are likely to plateau at a more effective level than manual text classification effectiveness levels.

Anti-Spam filtering has been the main focus of spam-related research, so the associated algorithms have been tested and evaluated many times through. In [1], the authors investigate a Naive Bayesian classifier to be used to filter spam emails from

legitimate emails. They're findings include that although the Naive Bayesian classifier has high spam recall and precision, it isn't viable when blocked messages are deleted, and additional safety nets are needed to make it have a significant positive contribution. These strict aspects of classification systems are important to developing the highest quality filters over time. In another paper [16], the authors recognize that there are several forms of Naive Bayes to consider. They compare the positive and negatives of 5 different versions of Naive Bayes. They find that when all things are considered, the multinomial Naive Bayes with Boolean attributes is superior. This version gave the best trade-off between ham and spam recall, and also had a lower computational complexity. The multinomial Naive Bayes classifier treats each message as a bag of tokens, eventually representing each message as a vector that reflects the presence of words in the message. Machine learning is crucial in the process of classifying spam, both in the traditional spam vs. anti-spam classification, and in the more sophisticated classification objective of this paper.

CHAPTER3

TOOLS & TECHNOLOGIES

3.1 AWS

We host our honeypots on Amazon Web Services (AWS). This allows us to have a consistent IP and maximize server ability.

3.2 Drupal

To set up our honeypot, we use the Drupal Content Management System (CMS) [5]. Drupal is quick to set up and provides a wide range of add-ons to allow for additional logging that proved helpful in our data collection. Also, its configurability is fine grained enough to maximize our availability to spam bots.

3.3 Python

Python is the only programming language we used for our data extraction and analysis. It has powerful libraries such as NLTK to assist in the type of analysis we are

interested in, and its file I/O is quick and easy to use. Additionally, its list comprehension capabilities proved to be essential in terms of readability and succinctness.

3.4 NLTK

One of the tools most relevant to our work is the Natural Language ToolKit (NLTK) for Python [4]. NLTK is full of tools that do much of the heavy lifting for a developer who wishes to use Natural Language Processing (NLP) on a set of textual data. NLTK comes with a suite of tools flush full of language processing programs. NLTK also includes a wide variety of corpora, spanning multiple languages. Some of these corpora are pre-tagged, and can be used in machine learning to train a classifier. Some of the corpora serve as useful content to practice using the language processing tools on. The corpora come in many flavors, spanning from classical texts, to political speeches, and even to internet chat room text. The corpora are extremely well organized and serve many purposes. It comes with easy to use graphing and display capabilities, which allows for a deep and unique understanding of language characteristics that a given language set possess. NLTK also comes with ready-to-run clustering algorithms, which are discussed later in this work. Along with NLTK, a free book is provided that not only teaches the user how to navigate around NLTK within Python, but also provides great insight into the basic principles and mechanisms of NLP as a whole. The aid that NLTK provides made many of the results in this paper plausible, and was essential in the understanding and processing of the data that we collected.

3.5 Alchemy

Alchemy is an API which takes a chunk of text and returns many useful features of it, using its own behind the scenes analysis tools. These features includes but is not

limited to extracting keywords and concepts, the taxonomy of the text, and proper nouns that otherwise would be difficult to extract. [14]

3.6 Google Language Detection

In order to programmatically detect what language a post was written in, we used a ported version Google's language detection. The API is also capable of detecting multiple languages. Analyzing posts in languages other than English is outside the scope of this paper, so we only considered posts in English for our data set and analysis.

3.7 SANS Internet Stormcast Center (ISC): Common Vulnerabilities and Exposure (CVE)

CVE is a system designed to provide structured data for information security vulnerabilities. Any security researcher can request a CVE number for a vulnerability found in a piece of software or product. For our use we used a public API provided by the Luxembourg CERT (CIRCL) which stores a publicly accessible CVE database. [15]

3.8 Stanford POS Tagger

In order to obtain the most accurate POS assignment to our blog spam, we used the Stanford Log-linear Part-Of-Speech Tagger. The tagger is a Java implementation of the tagger described in [26]. At the time this paper was published, the tagger resulted in a 97.24% accuracy on the Penn Treebank WSJ, 4.4% better than any

previous tagger of its class. This tagger achieved this accuracy based on the following ideas:

- Using both preceding and following tag contexts via a dependency network representation.
- Broad use of lexical features, including jointly conditioning on multiple consecutive words.
- Effective use of priors in conditional log-linear models.
- Fine-grained modeling of unknown word features.

3.9 Geolocation Tool

Cal Poly's Information Technology Services (ITS) department allows us to send in a list of IP addresses and in return get a list of the IP addresses paired with their geolocation. This helps us categorize the corpora and entities.

3.10 Megam

NLTK's built-in classifiers are highly functional, however they're primarily academic and not necessarily meant for heavy lifting. To get a higher performing classifier, we implemented MEGA Model Optimization Package (Megam) [7] into NLTK. Megam's performance was both more accurate and much faster.

3.11 Porter Stemmer

'Stemming' is the act of reducing words to their stems. There are many algorithms that stem words, and NLTK offers a few implementations. For our use, we use the Porter stemmer. The Porter stemmer is a common stemmer used in the normalization words within the NLP pipeline. [19]

CHAPTER4

EXPERIMENTAL DESIGN

The objective of our experiment is to gather a sample of blog, comment, and link spam that is representative of the internet as a whole, create corpora from the collected samples, and then analyze and draw conclusions from that corpora. One of our hypotheses is that we can identify a set of botnets that dominate the bulk of spam entries we receive, and then train a classifier that can accurately identify which botnet the spam comes from.

We want to make our sites as easily accessible to the spam bots as possible, while introducing controlled, calculated differences between the three of them. To do so, we set up an Apache server on an Amazon Web Server (AWS) virtual machine. Within the Apache server we enable virtual hosting, which allowed us to host the three websites on one Apache instance. Each website deployment used the same Content Management System (CMS), Drupal, with the same configuration, to give each site the same accessibility options so that the bots had the same capabilities within each site. In order to post content, a user has to be logged in. The bot has to go through the registration process, which includes email verification, in order to post content. Making the registration process slightly involved creates a sophistication threshold for the spambots who can interact with our sites. We do this in the hope of reducing

noise, so that only well formed spambots can post content.

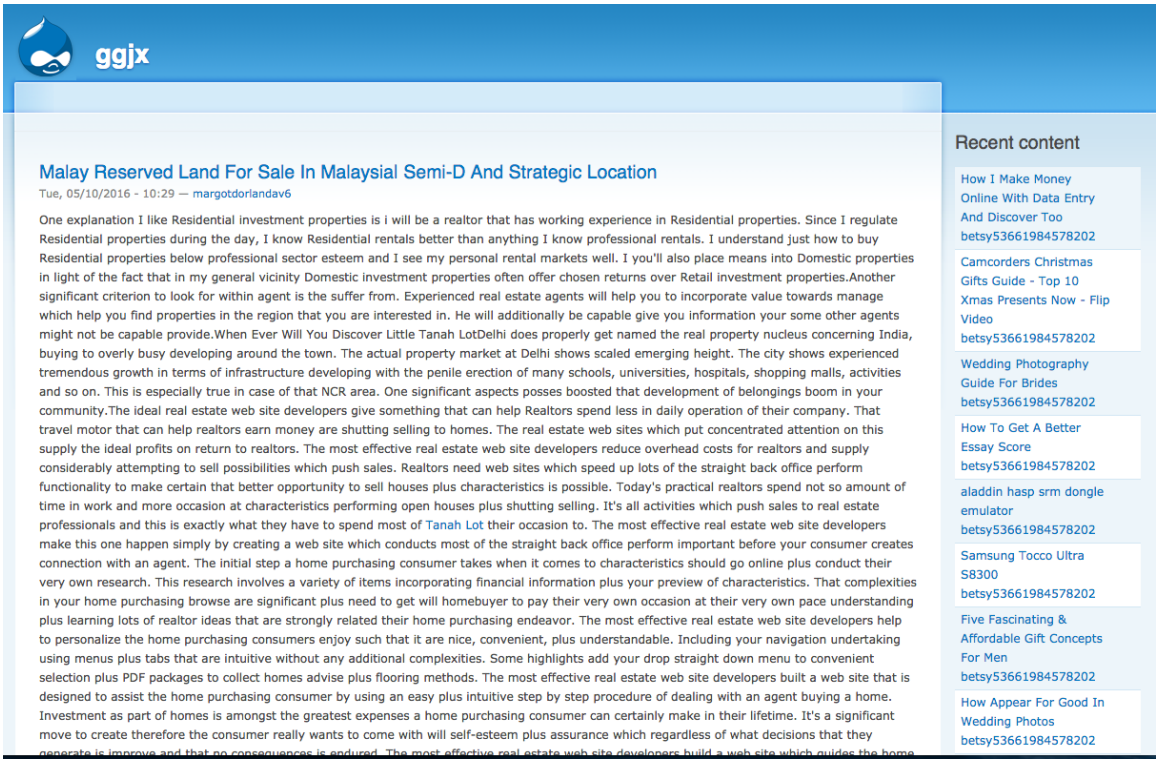


Figure 4.1: The ggjx honeypot as it appears to a human viewer.

The three honeypots we deploy are homogeneous. The only difference was the theme of the first post and the domain name. Ggjx.org is fashion themed, npcagent.com is sports themed, and gjams.com is pharmaceutical themed. Our motive behind the themes is to see if the spambots posted content similar to the theme of the website, or to detect any change in behavior from the spambots that we can attribute to a specific theme. The only way we 'theme' each website is by making the initial blog post. The bots can post content to our site in two ways: new blog posts, and comments on existing blog posts. Our content analysis is based solely on those two sources of content.

4.1 Data Collection

Within the CMS we enable some extra logging to enhance our feature set, such as IP tracking and number of logins. As can be seen in Figure 4.1, the layout of the honeypots is extremely rudimentary. Although our honeypots are online for a long period of time, the time period we consider in our data is restricted to nearly a month. We do this for a number of reasons. We want to have a sufficient amount of data to draw reliable conclusions from, however we want to limit the time period that we consider so that the data we observe is more likely to be representative of a single spam campaign per botnet. This way, when we isolate and analyze what we consider a single botnet entity, we observe its behavior only for a single instance of its spam lifecycle, as opposed to multiple campaigns that use the same botnet. Our intuition is that it would be easier to draw meaningful conclusions from a single run of a botnet rather than multiple runs. Our Apache server also logs all incoming network requests, as is the default for Apache installations. Our dataset consists of a combination of the MySQL entries and Apache logs.

4.2 Corpora Formation

To form our corpora we take the raw data from the MySQL database and Apache server, and divide it into three tables. Each of our three honeypots has its own set of tables, so the corpora consist of nine files. The three tables are defined as follows:

- User: The user table lists all of the users that were logged in the CMS. Each entry consists of the following attributes:

  uid: The unique user ID for this user.

username: The username that was registered with this account.

date_created: The date that the user registered at.

IPs: A list of IP / geolocation pairs that are associated with the account. These IPs are collected by the ip_tracker module in Drupal.

logins: The number of times the user logged in.

- Access: The Access table provides information about all of the HTTP requests logged by our Apache server. Each entry consists of the following attributes:

  uid: The unique identifier of the user who's accessing, if one is logged in.

  ip: The IP of the access request.

  request_path: The path specified in the HTTP request.

  request_type: The type of HTTP request, i.e. POST or GET.

  node_id: The Drupal node id requested in the HTTP request, if there is one.

  action: A keyword to describe what the access's purpose was. The set of keywords are described below:

    RESET: Request to reset a password.

    REGISTER: Request to register a new user.

    VIEW_NODE: Request to view a node, which node is filled into the 'node_id' field.

    ADD: Request to add content to the honeypot.

VIEW_USER: Request to view a user's profile, the user's id recorded into the 'node_id' field.

LOGIN: Request to login as a user.

PASSWORD: Request for the user's password, with request path 'user/-password/'.

EDIT_USER: Request to edit a user's profile.

OTHER: Uncategorized, didn't fall into any of the above keywords.

date: The date of the access.

- Content: The content table is where the blogs and comments from the Drupal websites can be found. Each entry consists of the following attributes:

  node_id: The unique node ID reflecting the node of the blog that this post was made on.

  author_id: An identifier corresponding to the 'uid' of the user who posted the content.

  date_publ: The date the post was made.

  hits: How many times the post has been visited.

  type: The type of post, i.e. a comment on an existing post or a new blog post.

  title: The title of the comment or blog post.

  text: The text of the comment or blog post.

  links: A list of link objects that are contained in the content of the post.

Each link object contains several attributes related to the link, explained below.

language: The language that the content of the post is in, extracted using the Google Language Detection API.

alchemy: The taxonomy of the post as reported by the Alchemy API.

- Link Object: A link object is contained within the 'links' attribute of each entry within the content table, explained above. Each link object contains the following attributes:

  url: The full URL of the link.

  surface_text: The text that a user would click on to access the link.

  domain: The base URL of the link.

  path: The path following the base url, i.e. '/user/4'.

  parameters: The parameters passed in through the url. i.e. '?ref=spammer&id=5'

  CVE: The 'Common Vulnerabilities & Exposure' categorization of the URL, if one exists, otherwise 'N/A'.

CHAPTER5

ANALYSIS AND RESULTS

5.1 Corpora

The immediate result of our experiment is the derivation of the corpora structure detailed in 4.2. The results contained in our corpora span from 16:00:00 November 25, 2015 to 15:02:31 January 5, 2016. The corpora are characterized in the following subsections.

5.1.1 User Table

To create a user, the spambot fills out Drupal's registration form and then activates the account through a verification email. Past the email, no sophistication is needed to create an account; no CAPTCHA or anti-spam measures are employed. We do this in order to get an accurate sampling of the different kinds of botnets present on the internet, including ones that are less mature. The general characteristics of the User

Table 5.1: General User Table Characteristics

Honeypot	Quantity	Average Logins Per User	Multinational Users	Number of Countries
ggjx	62992	1.066	0.443%	83
gjams	28230	1.102	0.488%	40
npcagent	34332	1.05	0.177%	53

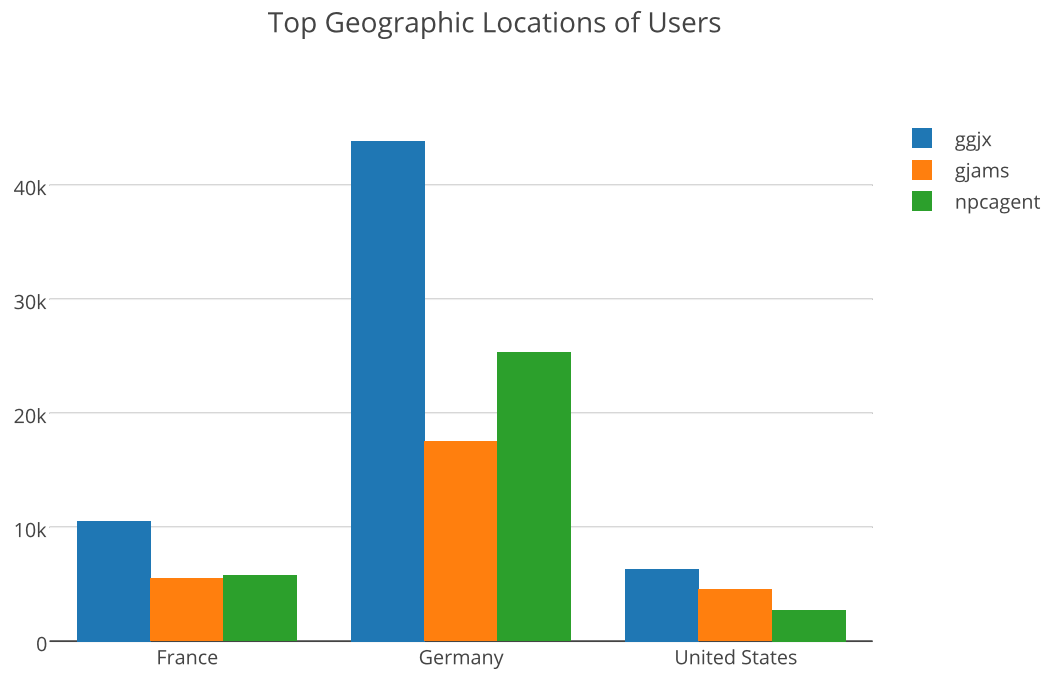


Figure 5.1: User Distribution for top geographic locations.

table can be seen in Table 5.1. It can be seen that the ggjx honeypot was by far the most popular honeypot in terms of user registration. It should be noted that gjams has a disadvantage, it was set up later than the other honeypots, and although it still had ample time to exist before the starting point of the experiment, its existence still had less time to propagate through site crawlers.

Another observation to be made is a caveat with virtual hosting. Apache's virtual hosting allows more than one website to be hosted from a single IP. However, only one website can be assigned to the IP, and so if a bot were to visit the IP directly it is taken to ggjx. This is a likely explanation for ggjx's popularity over the other two honeypots. Bots more often than not use direct IP addresses to search for a website rather than the domain name. The geographical distribution of our users is also a point of interest. Our honeypots are accessed by a wide range of countries. Figure

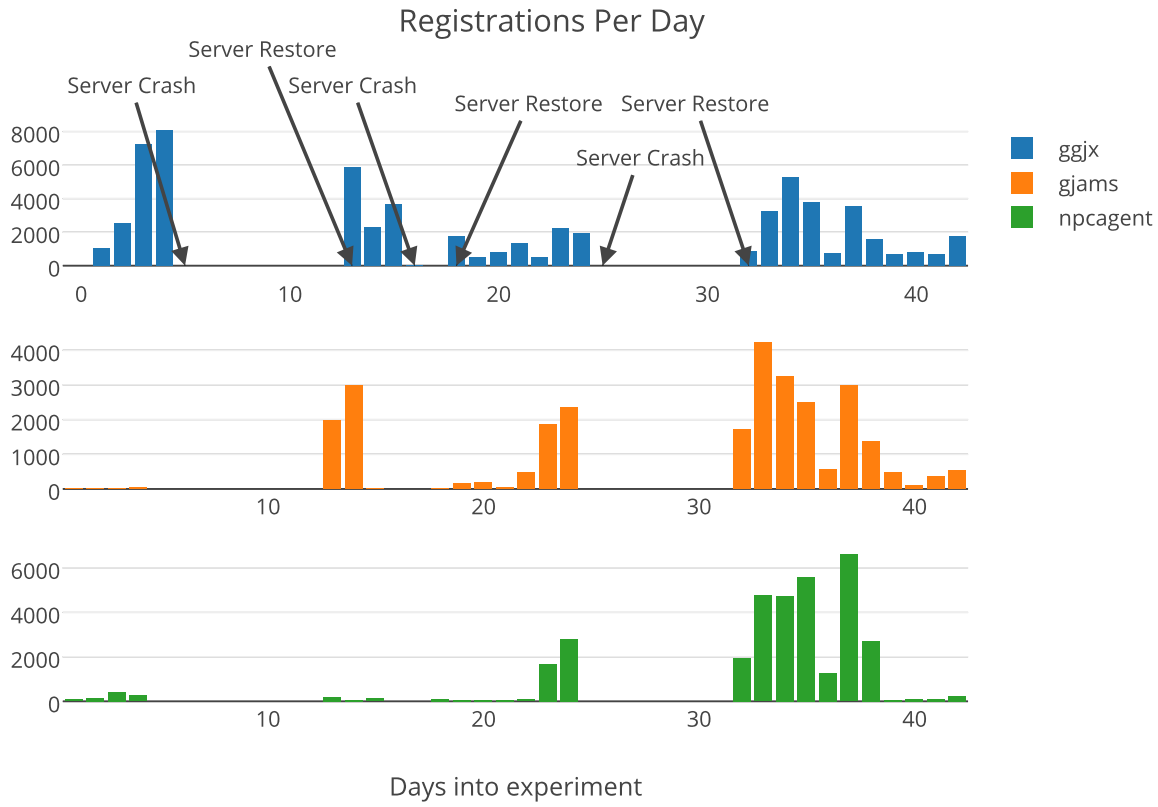


Figure 5.2: Registrations Per Day.

5.1 show the three most highly visited countries for the three honeypots: Germany, France, and the United States. Although there are visits from many countries, the majority of them are small in number. The geographic distribution amongst the three honeypots is similar.

Figure 5.2 shows the number of users that registered per day over the 42 days our experiment is run. We can see that ggjx again has a flatter registration slope, and near the end gjams and npcagent start spiking in registrations. It's also important to note the two dead zones shown in each of the registration figures. This is caused by overflows in the honeypot, the high quantity of user registrations overloaded the email daemon and it crashed, and therefore the spambots could not complete the verification step of their registration.

Table 5.2: Number of Access Requests per Honeypot

Honeypot	Quantity
ggjx	1100248
gjams	481138
npcagent	591238

5.1.2 Access Table

The Access table is based off of the Apache logs that were generated by the spambots sending network requests to our honeypot. Before they were added to the table, they were processed and some metadata was included to make it easier to characterize and analyze. This includes adding a few columns to assist in characterizing each entry. The general characteristics of the Access table can be seen in Table 5.2.

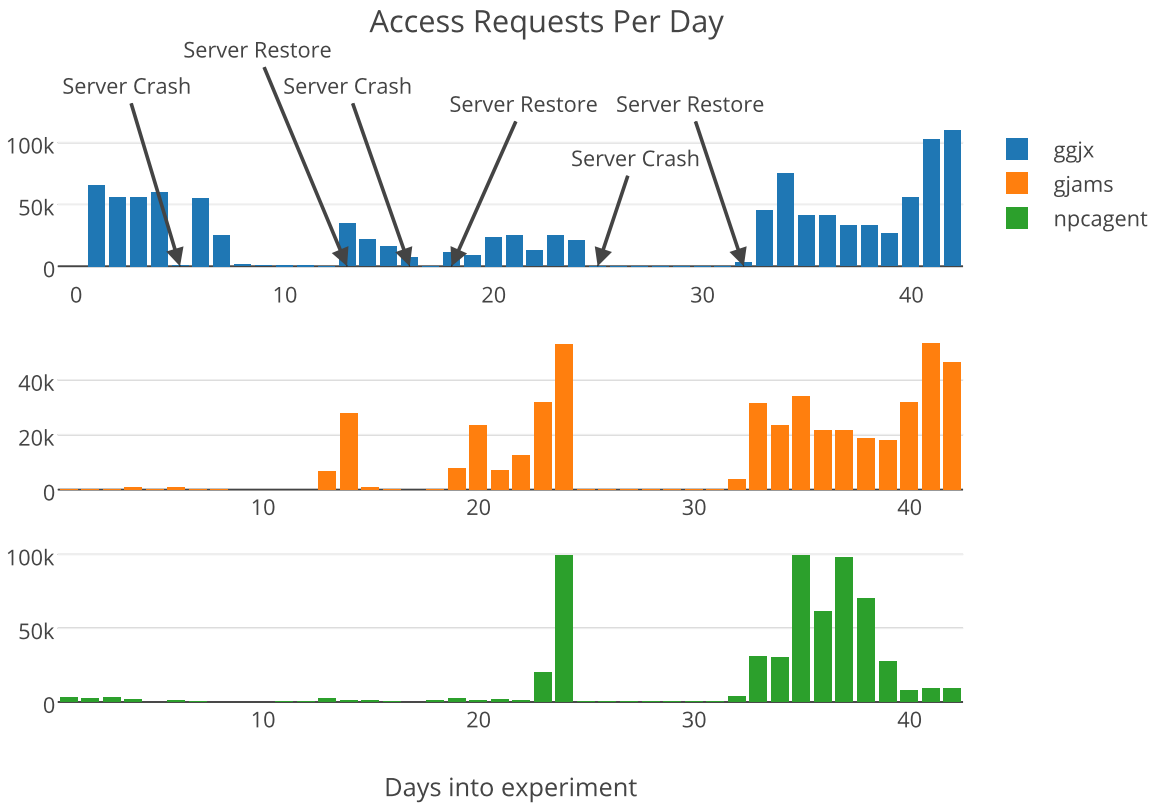


Figure 5.3: Access Requests Per Day.

The frequency of access requests for each honey pot is shown in Figure 5.3. Once again, we can see when the honeypot crashed under the weight of all the activity of the spambots. We can also see that the days leading to the crashes were traffic intensive, which makes sense. There is an enormous amount of access requests compared to the amount of posts and users. This means that a high percentage of the traffic our honeypots see is more likely passive scanners rather than intelligent spambots.

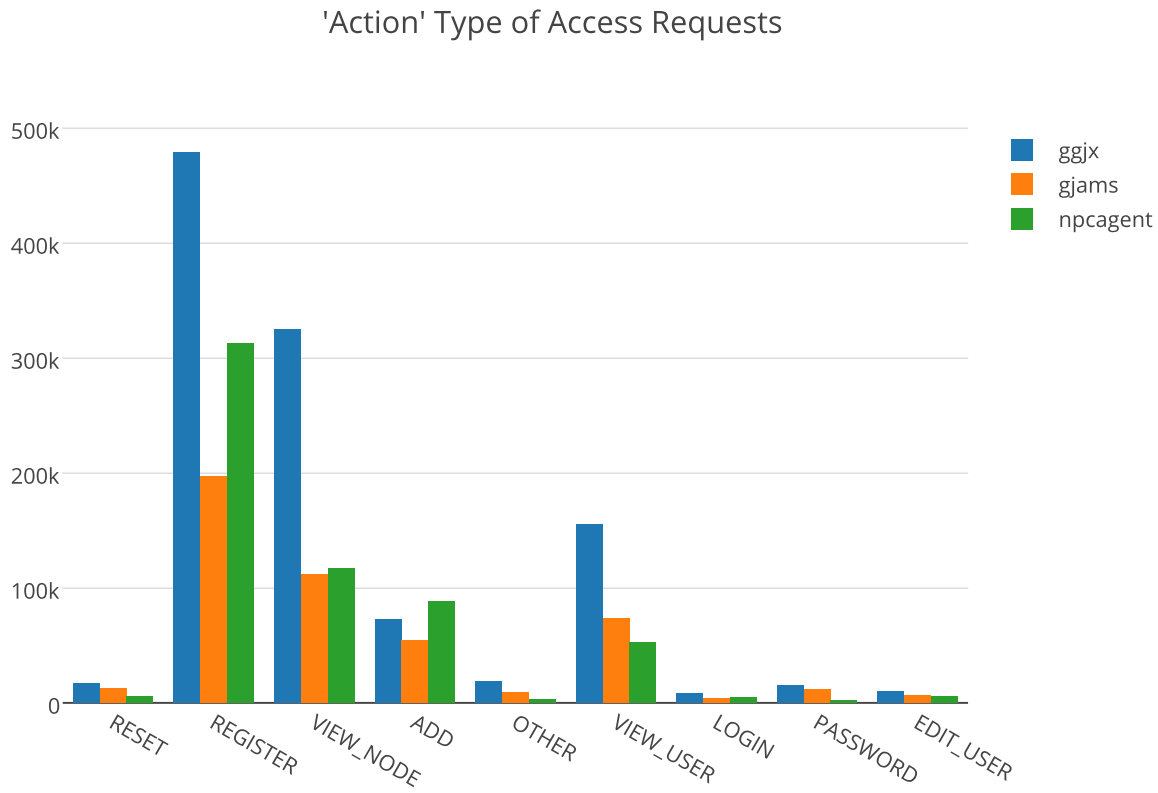


Figure 5.4: 'Action' type distribution.

In Figure 5.4, we can see the spread of action requests for each honeypot. We observe that most of the requests are trying to 'REGISTER' or 'VIEW_NODE'. A number of the access requests had random parameters in the URL, likely searching for some other common CMS such as Wordpress or Joomla, or searching for some specific vulnerability. These are categorized into the 'OTHER' category. For context, some

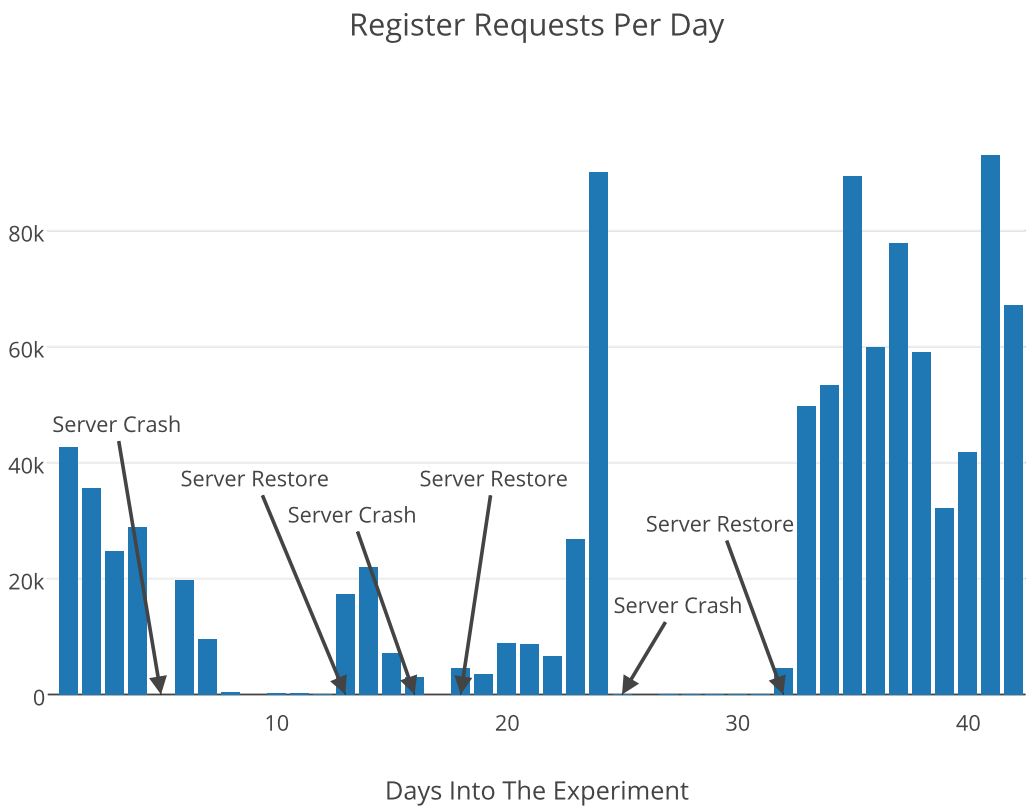


Figure 5.5: Register requests per day for all three honeypots.

examples of 'OTHER' requests can be seen in Figure 5.6. Finally, Figure 5.5 shows the combined 'REGISTER' requests from all three honeypots for the duration of the experiment. We can see that before each crash, the register requests are at a local maximum.

5.1.3 Content Table

The Content table contains all of the spam posts and spam comments that are made in the honeypots. It also contains some useful metadata that's pertinent to our experiment, as well as means to link them to the other tables in the corpora. General characteristics of the Content table can be seen in Table 5.3. There are some

```
'/about-us',
'/component/comprofiler/registers',
'/about-us',
'/terms-of-use',
'/song-catalog',
'/song-catalog',
'/artists',
'/forum',
'/tracker',
'/mobile/',
'/m/',
'/?view=contact',
'/about-us',
'/tracker',
'/admin/config?render=overlay',
'/admin/config/development/performance?render=overlay',
'/admin/config/development/performance?render=overlay&render=overlay',
'/admin/config/development/performance?render=overlay',
'/admin/config?render=overlay',
'/admin/config/development/performance?render=overlay',
'/admin/config/development/performance?render=overlay&render=overlay',
'/admin/config/development/performance?render=overlay',
```

Figure 5.6: Examples of 'OTHER' requests received.

Table 5.3: General Content Table Characteristics

Honeypot	Quantity	Average Hits	Average Links	Blog Posts	Comments	English Posts	Languages
ggjx	2279	28.237	2.356	1933	346	1962	13
gjams	2225	18.178	0.311	1686	519	2137	6
npcagent	1430	29.043	1.823	701	708	1409	6

interesting differences between the honeypots that we observe from Table 5.3. As expected, ggjx has the most content. Although gjams has less users than npcagent, it contains more content entries. This means that the users that interacted with the gjams honeypot were more active than the npcagent users. Conversely, each post in the npcagent honeypot was observed by other bots more than both gjams and ggjx. The posts in ggjx had the most links on average, and although gjams had the most active posters, only about 1/3 of them actually posted links in their spam posts. Another unexpected observation is that npcagent actually has more comments than posts.

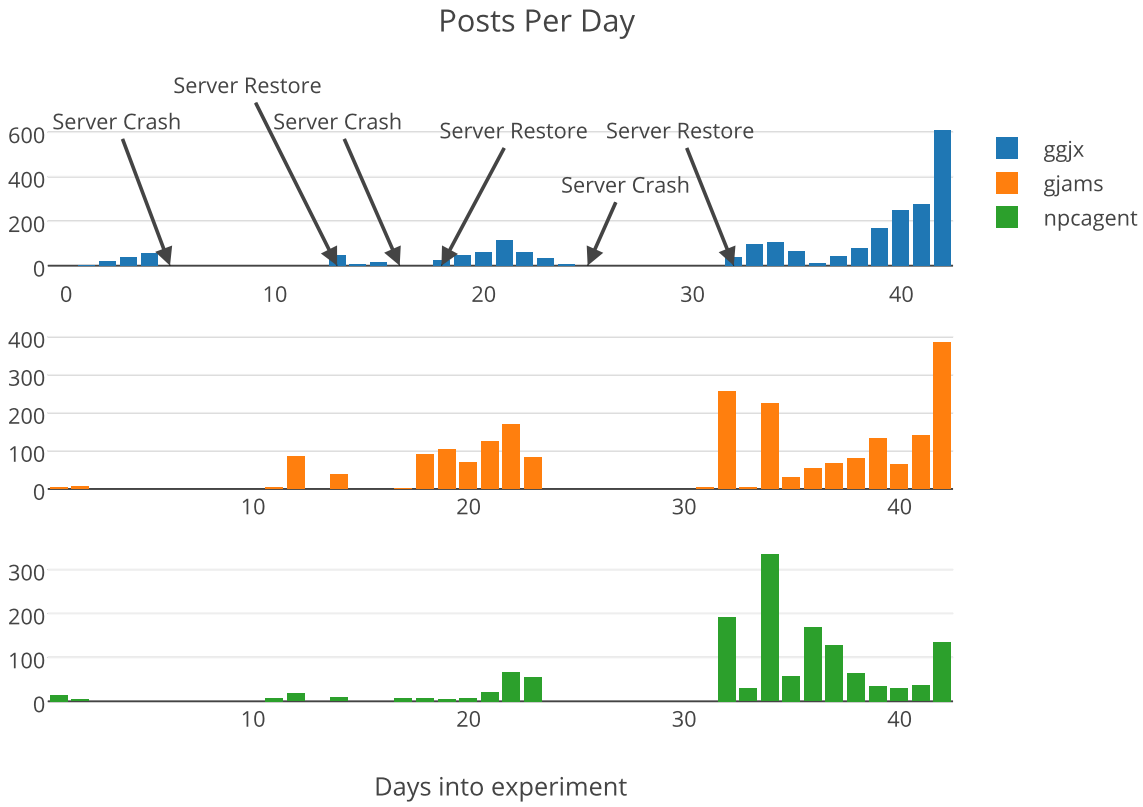


Figure 5.7: Posts Per Day.

Figure 5.7 shows the number of posts made per day for the three honeypots. The three honeypots don't seem to follow a trend, which suggests that the mechanism that controls when a bot will post is either complex or randomized. Again, we can see where the honeypots were down because of server overload when there were zero posts per day. When observing the characteristics of the Content tables, it's hard to draw correlations between the honeypots, unlike the User table where the three honeypots had similar characteristics. Using a number of analytical procedures on our corpora, we validate our hypothesis and gain additional insight into the mechanisms that drive spam bots. All of the results in the following subsections were derived directly from the corpora.

Table 5.4: General Entity Characteristics

Honeypot	Number Of Entities	% of Multinational Entities
ggjx	3332	4.862
gjams	1735	3.516
npcagent	1534	3.390

5.2 Entities

The use of spambots as the main delivery tool for spam content is widely accepted. Additionally, the use of 'botnets', or collections of these spambots, distributed over thousands and thousands of computers and managed by a central controller, is also common knowledge in spam research. A crux of our hypothesis is that each spambot, or 'user' in our context, who posted content in our honeypot is part of a larger botnet. We define these supposed botnets as 'entities', and use the terms interchangeably throughout this paper. One of the main ambitions of our analysis is to determine the plausibility of classifying a blog post into an entity, and to further determine a set of features that allow for that classification.

Each entity has the following attributes:

- id: A unique identifier for the entity.
- IPs: The set of IPs that are associated with the entity.
- usernames: The set of usernames that are associated with the entity.
- user_ids: A list of user IDs that are associated with the entity.

We construct the entities from our corpora programmatically. Each honeypot has its own respective set of entities. We form our entities with the assumption that botnets are connected. This means that every agent in a botnet is connected to every other agent in the botnet; if there is any direct or indirect connection between users we consider them in the same botnet. To form the entities, we scan through the list of

Table 5.5: Entity User Characteristics

Honeypot	Average Users per Entity	Maximum Users	Standard Deviation of User Quantity
ggjx	18.905	38001	682.704
gjams	16.271	14249	365.326
npcagent	22.381	23577	617.5

Table 5.6: Entity IP Characteristics

Honeypot	Average IPs per Entity	Maximum IPs	Standard Deviation of IP Quantity
ggjx	1.628	126	4.648
gjams	1.812	172	6.364
npcagent	1.414	76	2.849

the honeypots users, assigning each one to the entity it belongs to, or creating a new one if it doesn't match with an existing entity. The rules for entity construction as each user is scanned are as follows:

- If an entity exists that contains the username or the IP of the user being considered, the username, IP, and ID of the user is added to that entity.
- If more than one entity matches the above criteria, then all matching entities are merged into a single entity.
- If no entity matches the above criteria, a new entity is created with the username, IP, and id of the considered user as the initial data within the entity.

The results of our entity formation confirmed what is well known - the vast majority of the users that were interacting with our honeypots were interrelated from a network standpoint. It also suggests a significant finding: Each botnet has a shared intelligence, that is, a users login information is shared among many of the botnet's endpoints. The characteristics of our entity set can be seen in Tables 5.4, 5.5, 5.6, and 5.7.

The frequency distribution of posts per entity is a point of interest. The honeypot with the least amount of posts and activity, npcagent, also contains the entity with the most posts, by a significant margin. Also, the honeypot with the most activity,

Table 5.7: Entity Post Characteristics

Honeypot	Average Posts	Maximum Posts	Standard Deviation of Posts	% Of Entities Who Posted
ggjx	0.684	163	5.461	14.406
gjams	1.282	484	14.889	11.470
npcagent	0.926	664	17.449	12.712

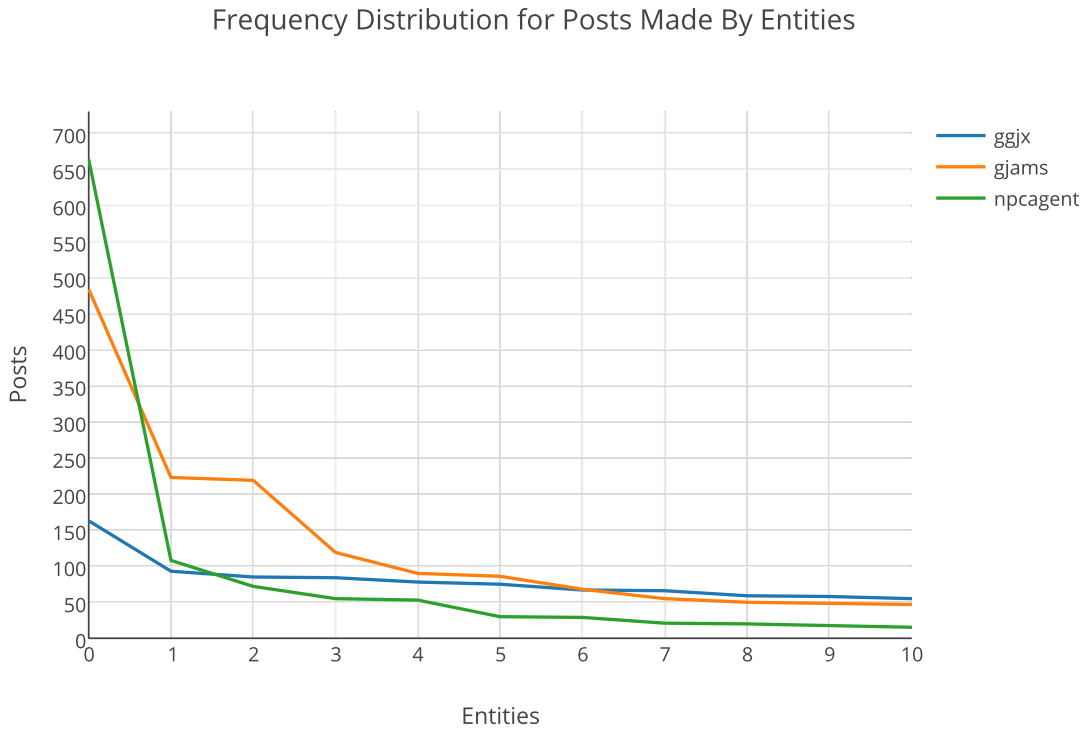


Figure 5.8: Post Frequency Distribution.

ggjx, has the flattest post frequency distribution. Ggjx's top poster contributed the least of the three honeypots' top posters. A small subset of entities is responsible for the majority of the posts, as can be seen in Figure 5.9. The more popular our honeypot is, the more evenly distributed the post volume is across entities, as can be seen by observing Figure 5.8.

To further compare the different characteristics of the entities that interacted with our site, we take the frequency distribution of the quantity of IP addresses associated with each entity. For consistency, the entities are sorted by number of posts in descending order, the same sorting used for Figure 5.8. Figure 5.10 shows the IP frequency

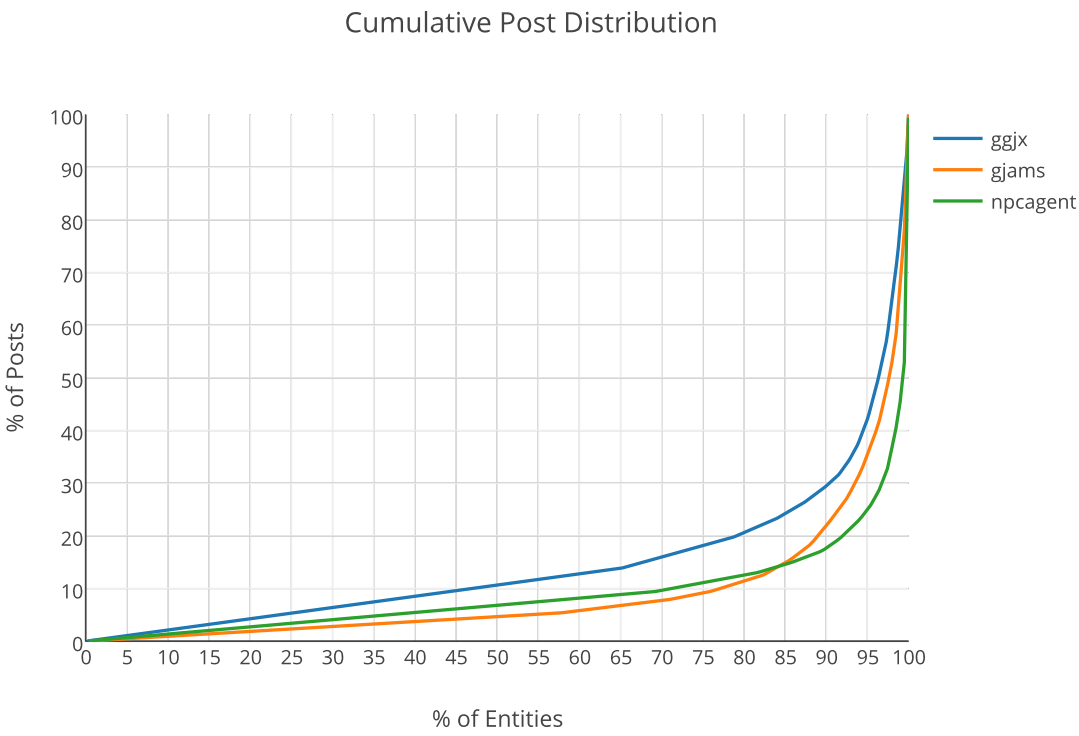


Figure 5.9: Cumulative Post Distribution when only considering entities who posted at least once.

distribution for each of our honeypots. We postulate from the post and IP frequency distribution graphs that each supposed botnet has its own unique characteristics. Although they are correlated, an entities quantity of associated IPs does not determine how active it is in posting content, and vice versa. Some entities were more active with a low number of IPs, and some entities had a large IP pool but didn't post much content. Overall, the entities that interacted with our honeypots had various sizes and some were clearly more developed than others. If our sample set is indicative of the overall internet space, our entity distribution suggests that the vast majority of spam posts come from a relatively small number of entities.

The entities we form from the corpora are paramount to the rest of the analysis we perform. As explained in the following subsections, it allows us to form a model of the distributed systems that interact with our honeypots during the time frame of

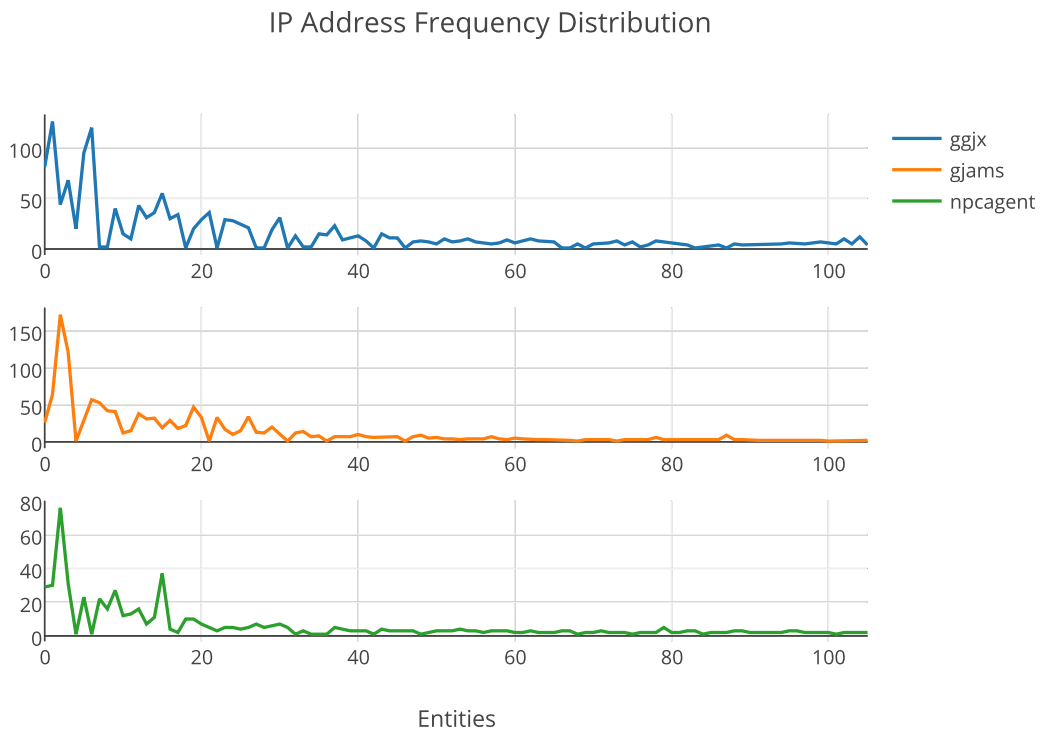


Figure 5.10: IP Frequency Distribution.

the experiment.

5.3 Content Analysis

The first leg of our analysis consists of using NLP to form feature sets from the documents found in the Content table of our corpora, and then using ML algorithms to gauge the significance of each attribute set. The main objective within the content portion of our analysis is to identify the traits within a post that are the most unique to its author.

5.3.1 Feature Sets

To accomplish our objective of finding the most distinguishing traits of spambots, we must first create a number of feature sets to model the document based off a particular trait. The different feature sets we derive are explained in the following subsections.

5.3.1.1 Bag Of Words

The first feature set we examined is a classic within NLP: Bag Of Words (BoW). BoW is a way to model the lexical content of a document. Put simply, each word in a document is put into a 'bag'. The document's ordering and syntactic structure is shed, and what's left is the set and count of words present in the document. BoW is easily understood by example, if a sentence X is 'The dog ran to the river to get the ball.' X's BoW representation would be: 'the': 3, 'dog': 1, 'ran': 1, 'to': 2, 'river': 1, 'get': 1, 'ball': 1. This representation suffices to capture the lexical content present in a document. In the context of the entities we analyze, the BoW model we create is slightly more complex. We first put the text of each document through what's known in NLP as 'preprocessing'. To start, not all words within a document are useful. The word 'is' and 'the' do not denote any sort of significance to a given text. In fact, if included in a BoW feature set, these small 'helper' words only clutter up the feature set, and actually damage its usefulness. Put another way, these words, known as 'stopwords', only add noise to the feature set, decreasing the BoW's signal-to-noise ratio. Deciding which words to exclude from a BoW is part of the art of creating a feature set. For our purpose, we excluded any stopwords (according to NLTK), any words less than 4 characters long, and any words that had any nonalpha characters in it. Another concept that is useful for increasing the effectiveness of the BoW feature set is stemming. Stemming takes a word and reduces it to its core

component. Put another way, stemming is a technique used to gain morphological unification within a set of words. For example, the word 'swimming' and the word 'swim' would be represented as different entities in an unaltered BoW implementation. This is harmful to the BoW's purpose, we're not concerned what form the root word 'swim' comes in, we just want to know that the word 'swim' is being used. When passed through a stemmer, the word 'swimming' would return as 'swim', and the BoW would be properly treat the two words as the same. Additionally, to normalize a given document with all the documents that it's being evaluated against, the 'bag' within the BoW needs to be standardized. To accomplish this, the union of the words in each document is taken to create a bag that contains every word present in the collection of documents. The resulting bag is generally much larger than the content of any one document, and so any given document's bag of words will highly resemble a sparse array.

5.3.1.2 Alchemy Taxonomy

Another group of attributes we used as a feature set is a result of IBM Watson's AlchemyAPI. IBM offers an API to submit text to, to which a set of relevant data is returned. In particular, it offers a 'taxonomy' evaluation. We submit the text of each document to the API, and it returns a list of taxonomies for the submitted text. The returned list is a list of pairs, the first item being the label of the taxonomy, and the second being Alchemy's confidence of the label rated between 0 and 1. If the confidence is low enough, it also includes a 'confident': 'no' attribute. For the purpose of our analysis, we discard any unconfident labels. We then strip the confidence value of each taxonomy listing for the purpose of counting, so any 'confident' taxonomy reported by Alchemy API is counted equally.

5.3.1.3 Link

A particularly interesting set of attributes that we used as a feature set was the link content within each document. As discussed in 2, it's widely accepted that one of the main goals of the spam bots that we observe is to promote the value of other websites as seen by search engines. To see how pertinent links are in distinguishing between entities, we created a feature set that contains the links within the document. To create the feature set, each document is parsed for any HTTP links. Each link is stripped down to just its core domain name and then added to the feature set. For example, 'http://www.example.net/?ref=5&source=spam' would be recognized, stripped down to 'example.net' and added to the feature set.

5.3.1.4 Vocab

A simple metric we use is the normalized vocabulary size of a document. The equation to get this is straightforward, the number of words in the set of words in a document divided by the total number of words in the document. The feature set of only Vocab only has one element in it; the aforementioned ratio. This metric is useless for Naive Bayes, since we use the Boolean modification. Unless it's an empty document, the feature set would be the same for every document.

5.3.1.5 PoS

Another feature set we use is the PoS makeup of a document. To achieve this, we first pass the text from each document into a PoS tagger, in our case the Stanford PoS Tagger as mentioned in 3. the text of the document is converted into a list of words and passed to the tagger, and the tagger returns a list of pairs. Each pair consists of the originally passed in word and the PoS that the tagger identified for

that word. From there, the original text is discarded and the PoS of each word are added to the feature set. The PoS feature set is useful because it abstracts away the actual words used in the document, leaving the syntactic structure behind. This can lead to correlations that otherwise would be difficult to draw. For example, the words 'beautiful' and 'pretty' are synonyms in most cases, however they would be viewed as completely separate in a BoW feature set. When considering their PoS, however, both would come back as 'adjectives' and therefore be considered the same.

5.3.1.6 N-Grams

The previously listed feature sets provide different ways of modeling a document in order to isolate specific attributes of the document. 'N-gram' is not a standalone model like the previously mentioned feature sets, but rather a way of modifying an existing model to change how effective it is at whatever it's trying to represent. An 'n-gram' is the contiguous sequence of n items in a given sequence. In our case, it's every n-length sequence of words that appear in a document. For example, the bigrams of the sentence, 'The dog ran to the river to get the ball.' is: 'The dog', 'dog ran', 'ran to', 'to the', 'the river', 'river to', 'to get', 'get the', 'the ball'. For our use, we consider bigrams (2-grams), and trigrams (3-grams). In some cases, such as the Link feature set, applying an N-gram modification does not make sense, because the links usually do not fall next to each other, and the order that the links appear in the document doesn't matter to us. In other cases, such as the PoS feature set, N-gram modification makes much more sense. PoS by itself doesn't portray the syntactical structure of a sentence very well. But, with an N-gram modification, the PoS model is much more representative of the actual syntactic structure of the document.

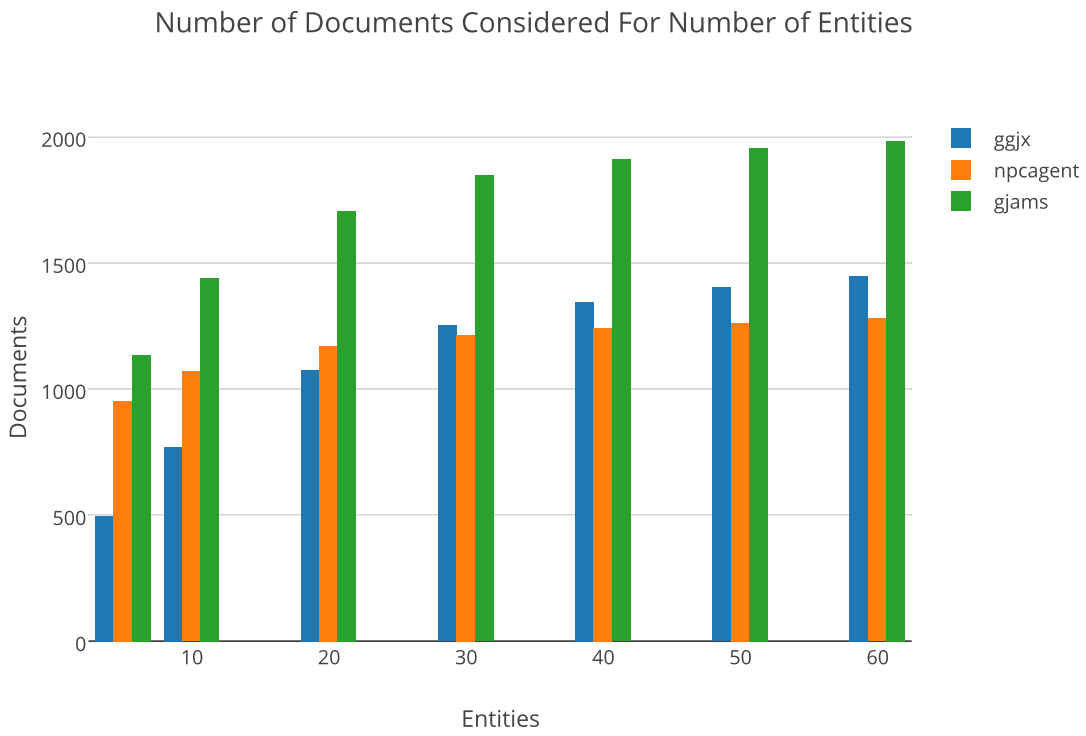


Figure 5.11: Number of Documents per Group of Entities.

5.3.2 Algorithmic Classification Analysis

In the following two subsections we test our feature sets with the Naive Bayes and Maximum Entropy classification algorithms. We evaluate each algorithms effectiveness at predicting the posting entity's ID based on the provided feature set. Before we do so, we observe the feature sets used in each classification instance. With each feature set, we must divide it into a 'training set' and a 'test set'. We use an 80/20 split, with 80% of the feature set assigned to the training set, and 20% of the feature set assigned to the test set. The number of documents considered varies based on how many entities we consider. We consider the entities in descending order of how many documents they've posted. The number of documents considered based on entities considered can be seen in Figure 5.11.

Table 5.8: An example of a Boolean modification of a feature set.

Before Boolean Modification	After Boolean Modification
'the': 3, 'dog': 5, 'plays': 0	'the': 1, 'dog': 1, 'plays': 0

Table 5.9: An example of a normalization modification of a feature set.

Before Normalization Modification	After Normalization Modification
'the': 3, 'dog': 5, 'plays': 0	'the': .375, 'dog': .625, 'plays': 0

Due to the differences between the Maximum Entropy and Naive Bayes classifier, it's logical to modify our feature sets to maximize efficiency for each algorithm. We create two modifications for our feature sets: Boolean and normalization. Their descriptions are as follows:

- **Boolean:** The Boolean modification simplifies the attributes within a feature set to binary values. This results in the loss of multiplicity, and is a more simple representation of the attributes within a feature set. An example of a Boolean modification can be seen in Table 5.8.

- **Normalization:** The normalization modification normalizes the value of each attribute within a feature set. This effectively makes each feature set independent of the length of the document that it's modeling, allowing longer documents to be more easily compared to shorter documents. To perform this modification, each attribute's value within a feature set is divided by the total of all of the attribute values within the feature set. An example of a normalization modification can be seen in Table 5.9.

We compare the effectiveness of our two feature set modifications by using each modification with both algorithms using the BoW feature set. In Figure 5.12, we see the results of using the Boolean modification. The two algorithms perform similarly in this case, with Maximum Entropy slightly outperforming Naive Bayes. Figure 5.13 shows the two algorithms performances with the normalization modification. The Maximum Entropy classifier performs similarly, while Naive Bayes performs dramat-

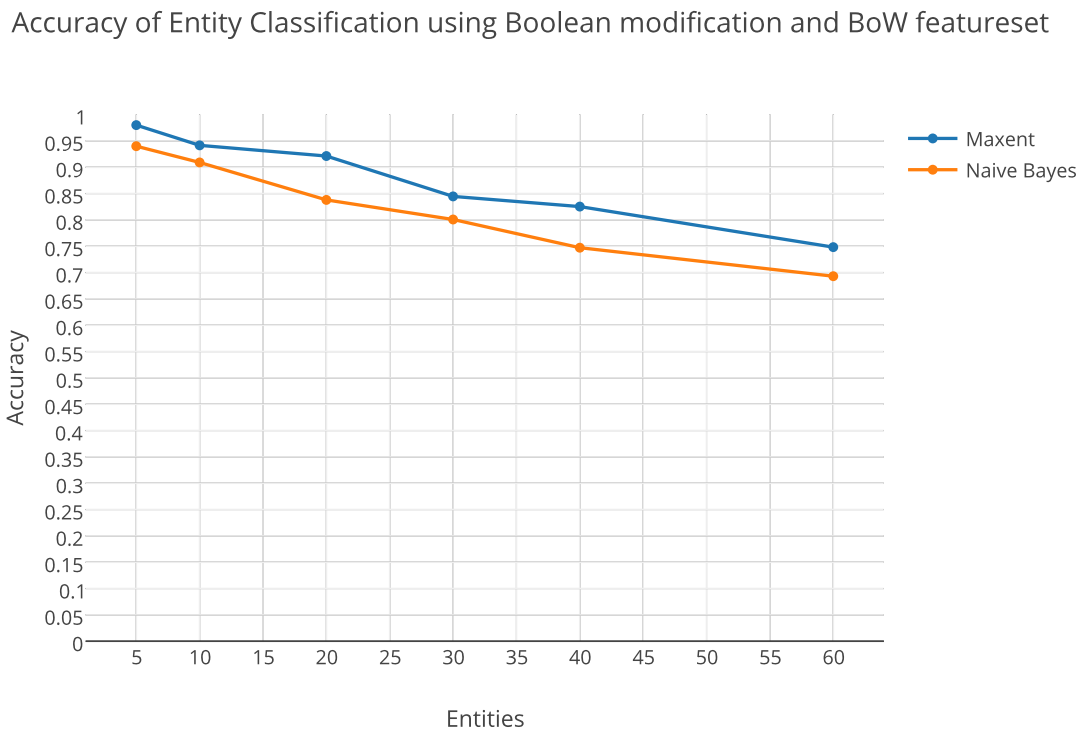


Figure 5.12: Prediction Accuracy With the Boolean feature set modification.

ically worse. This is expected, because Naive Bayes treats each value as a discrete value with no relation to any other values, whereas Maximum Entropy doesn't. Put another way, Naive Bayes doesn't take into account that '.8' is very close to '.79', whereas Maximum Entropy does. It's obvious that the Boolean modification is essential for an effective Naive Bayes classifier. In this paper it's assumed that any analysis using a Naive Bayes classifier has the Boolean modification applied to its feature set. We can also see that Maximum Entropy out performs Naive Bayes in both cases.

When cross examining Figure 5.12 and 5.13, it's unclear which modification scheme is best for Maximum Entropy. To get a better idea of which modification is best for Maximum Entropy, we use both modification schemes with the Maximum Entropy classifier again, only this time we use a different feature set: Link. The results can be

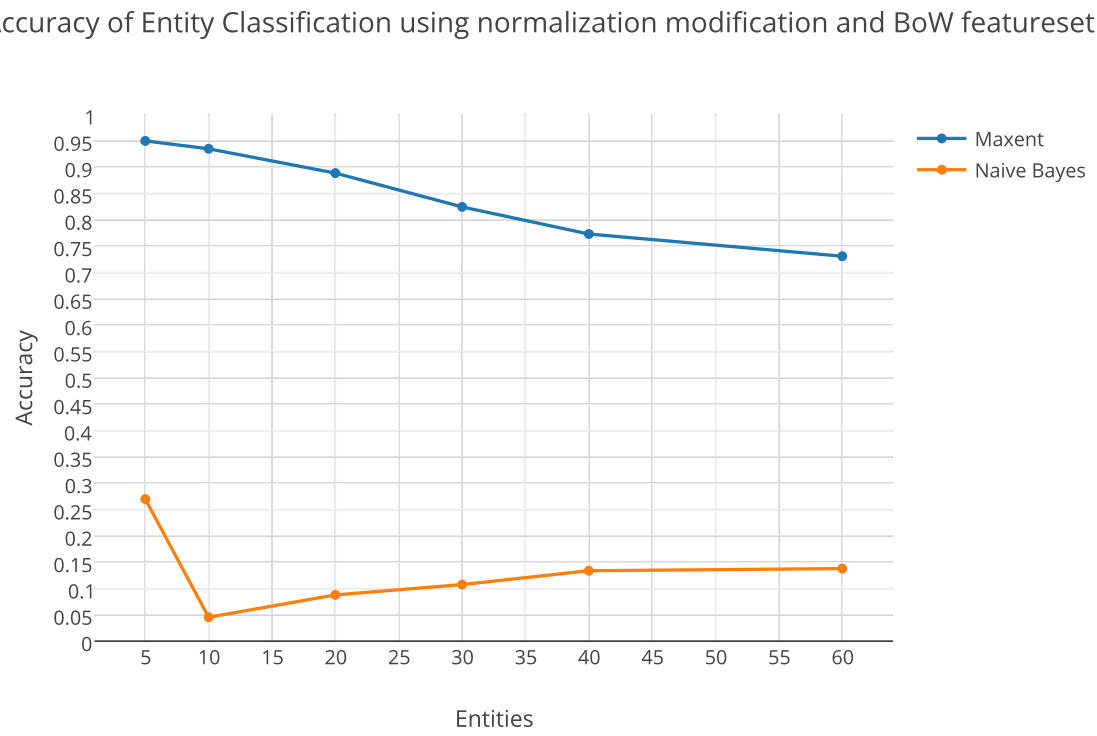


Figure 5.13: Prediction Accuracy With the normalization feature set modification.

seen in Figure 5.14. We see that the normalization modification slightly out performs the Boolean modification. Therefore, in this paper, it's assumed that any analysis using a Maximum Entropy classifier has the normalization modification applied to its feature set.

In addition to feature set modifications, we also employ a word cutoff. When using BoW, we only want to include the most common words. This limits the size of our feature set which increases the classification speed, and cuts off lesser used words which can end up harming performance. To find the optimal cut off point, we graph the accuracy of our two algorithms with the BoW feature set and a varying top word cutoff threshold. The resulting graphs can be seen in Figures 5.15 and 5.16. We observe that the cutoff threshold doesn't effect the accuracy very much. This goes to show that the majority of information in BoW is gathered within the most common

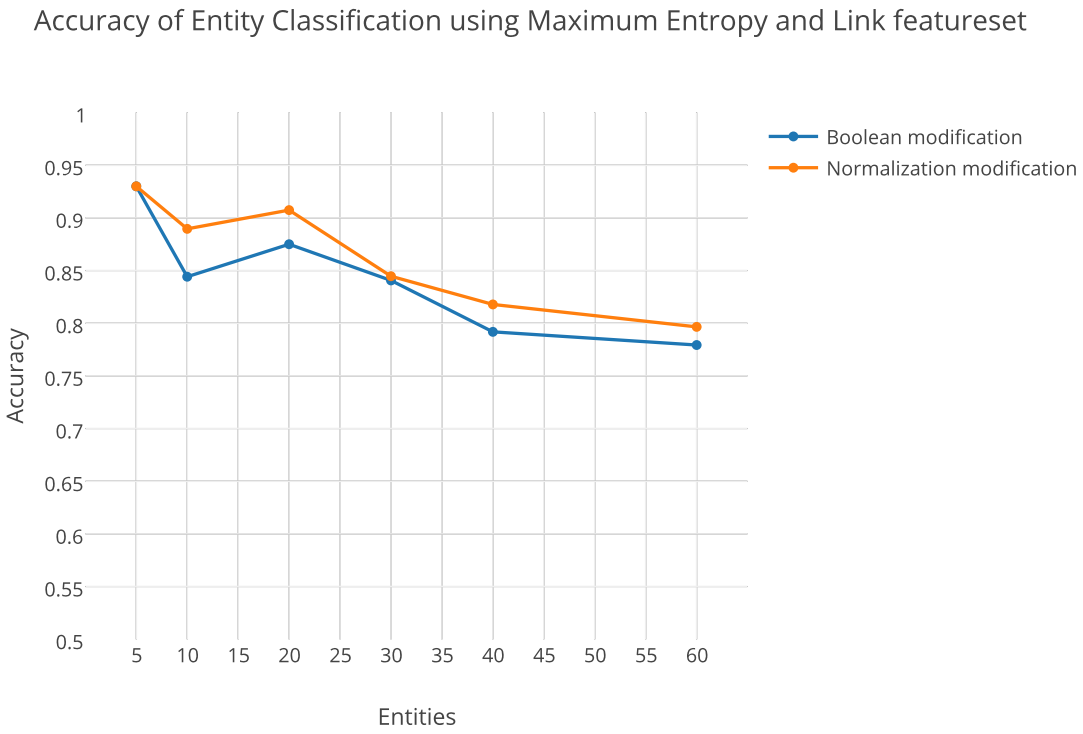


Figure 5.14: Prediction Accuracy of the Maximum Entropy classifier with the two feature set modifications.

words, and the least common words don't add any effective data to the feature set. For the rest of the paper, we use 30% as the top word cutoff threshold.

5.3.3 Maximum Entropy Classification

The first classification algorithm we use for NLP is the Maximum Entropy Classifier. The Maximum Entropy classifier is a conditional classifier. This means that the classifier can predict how likely a label is, but it must be provided an input set of features. This limits the applications that the classifier can be used for. Although its functionality is limited when compared to a generative classifier like Naive Bayes, the tradeoff is that the Maximum Entropy classifier outperforms the Naive Bayes classifier in some scenarios. Internally, the main difference between a Maximum Entropy

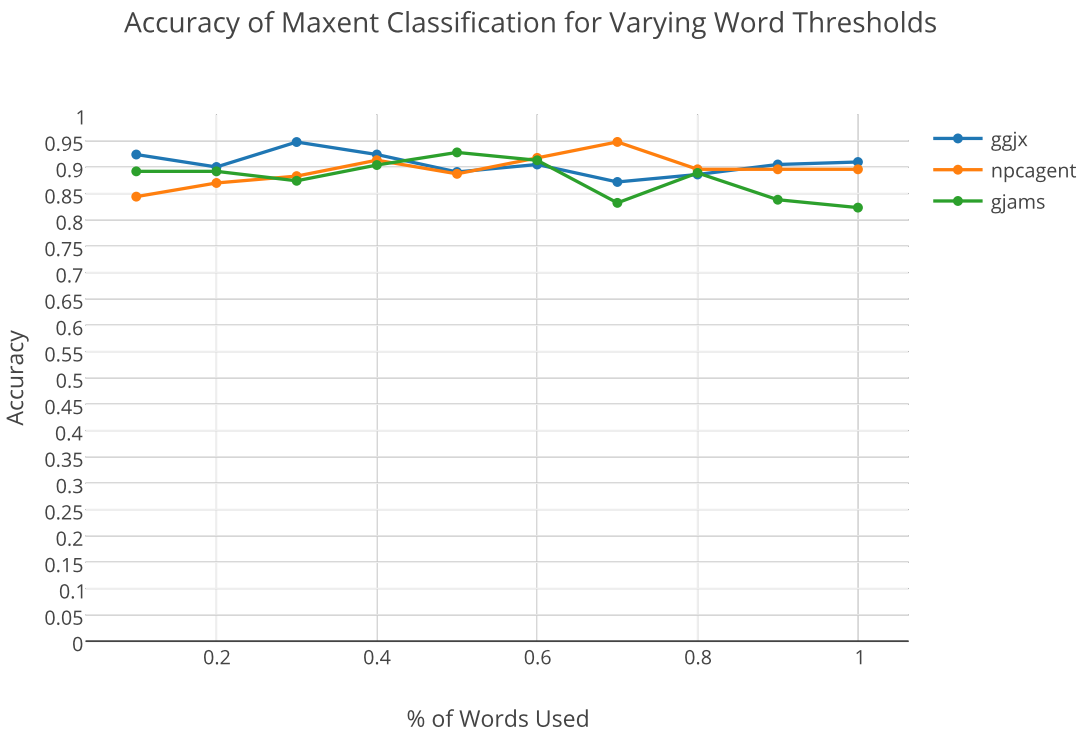


Figure 5.15: Accuracy of Megam classifier with a varying top word Threshold.

classifier and a Naive Bayes classifier is that the Naive Bayes classifier assumes all features to be weighed equal. This leaves it vulnerable to the effects of commonly occurring feature set patterns such as double counting. Maximum Entropy doesn't make this assumption, instead it iterates through the attributes within a feature set and finds which ones give the most information gain, and then weighs those attributes more heavily.

There are some additional slight differences we employ in feature sets between Maximum Entropy and Naive Bayes. For one, when using the BoW feature set, a smaller universe of words are considered when creating the 'bag'. To shrink the feature set to more relative words, we only use words that occur in mostly all of the considered documents; a statistical quasi-intersection of the documents, as opposed to the statistical union of the documents used in the BoW in Naive Bayes. This approach

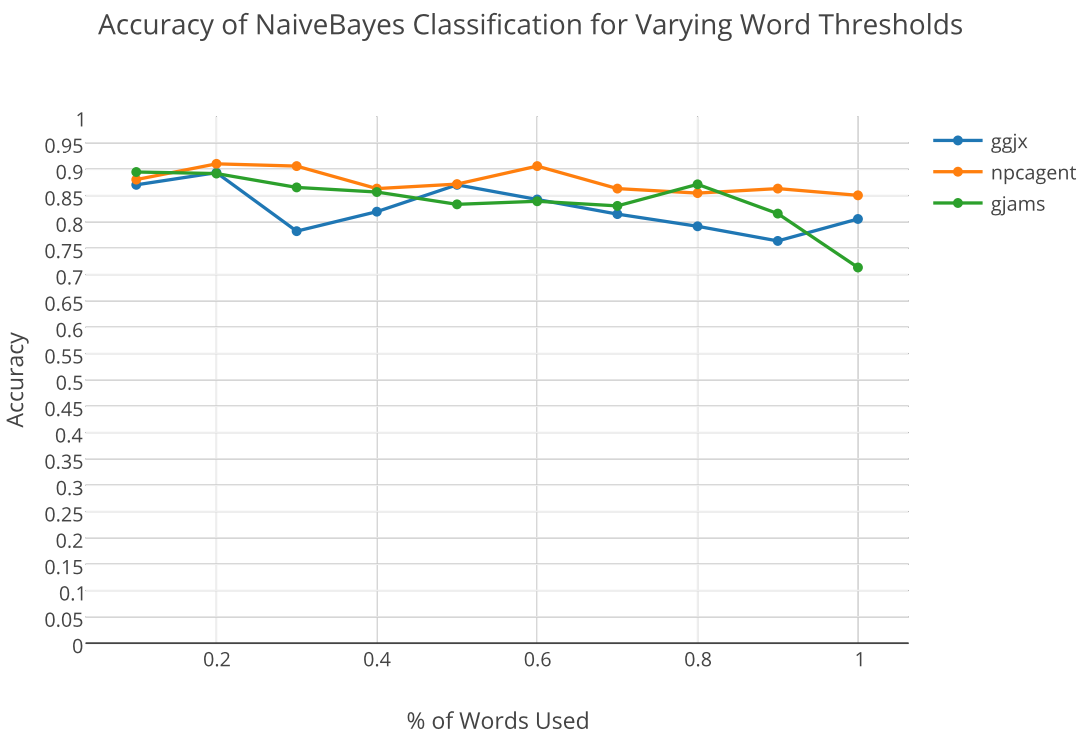


Figure 5.16: Accuracy of Entity Classification using Naive Bayes with a varying top word threshold.

wouldn't work with Naive Bayes, because every word in the feature set would be 'True'. As mentioned in 5.3.1, the feature set when working with Maximum Entropy is a set of normalized weights as opposed to the set of Boolean values used with Naive Bayes.

The Maximum Entropy implementation we use is megam [7], which NLTK provides an interface for. We create the feature sets described in 5.3.1, and then use it to train a Maximum Entropy classifier. From there, we evaluate the accuracy of the classifier and finally the effectiveness of our feature sets.

In figure 5.17, we use the classic BoW feature set. The results are conclusive. When only considering the top five contributing entities, the classification accuracy for npcagent is 100%, and greater than 95% for the other two honeypots. This feature set

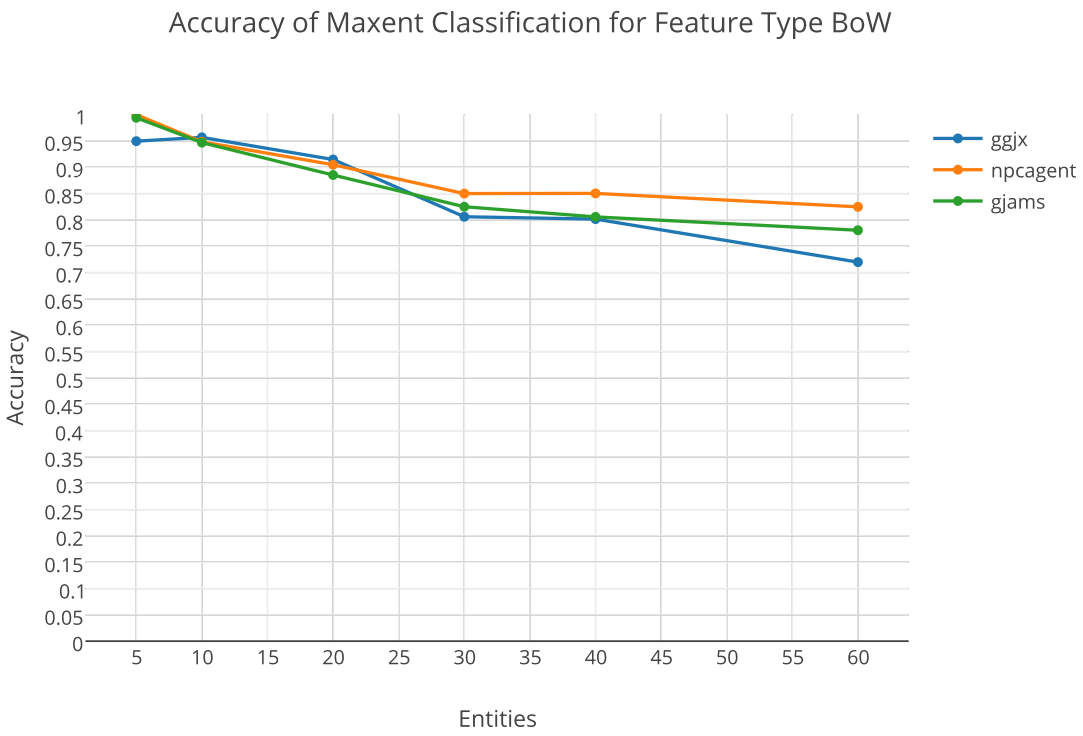


Figure 5.17: Accuracy of Megam classifier with the BoW feature set.

offers the highest prediction accuracy for our Maximum Entropy classifier, and it shows that considering the particular words used within a document is the most reliable way for a classifier to distinguish between entities.

In Figure 5.18, we train the Megam classifier with the Alchemy Taxonomy feature set. The results are impressive, and are a testament to the accuracy of AlchemyAPI's ability to provide an accurate taxonomy of a text. Aside from that, it also provides an interesting distinction. The accuracy that the Alchemy Taxonomy feature set provides suggests that within a given entity the posts that the individual spambots are posting have similar themes. Relating back to the tools discussed in 2.1, this distinction suggests that the spambots are all running XRumer or some similar text generation program, and that they select words from a similar pool of vocabulary for their text generation. It also supports a recurring implication that the semantic

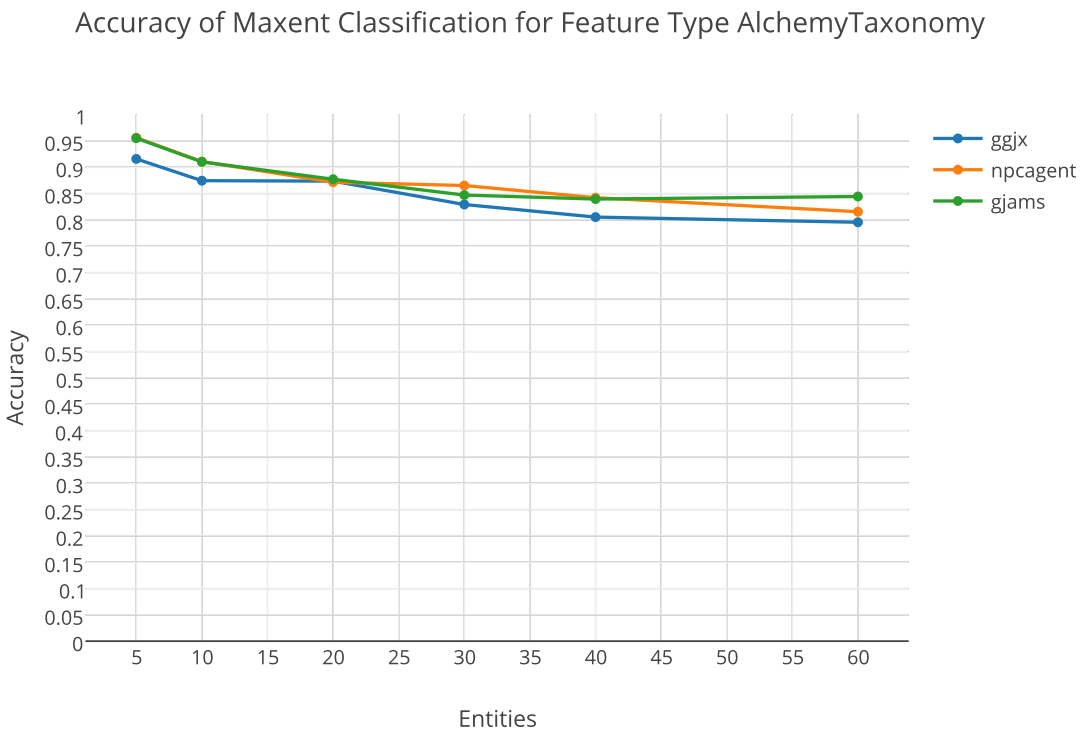


Figure 5.18: Accuracy of Megam classifier with the Alchemy Taxonomy feature set.

content of a post is the most reliable way to distinguish the post between spam bots. Also, the Alchemy Taxonomy feature set size is much smaller than the BoW feature set, and it still results in high classification accuracy. This suggests that the full BoW feature set isn't necessary for near perfect classification, and perhaps a proper combination of a smaller BoW and taxonomy would suffice.

In Figure 5.19, we evaluate the Link feature set with the Megam classifier. The results for each honeypot varied. The Link feature set proved to be a reliable feature set for ggjx, but it was poor for gjams, and less effective with npcagent. These results correlate with Table 5.3. As intuition suggests, the more scarce links are within posts the less reliable they are for classification. We can still draw some conclusions, namely that if links are present in many posts, they're a reliable way to distinguish between botnets. Also, it reinforces the claim that the individual components that

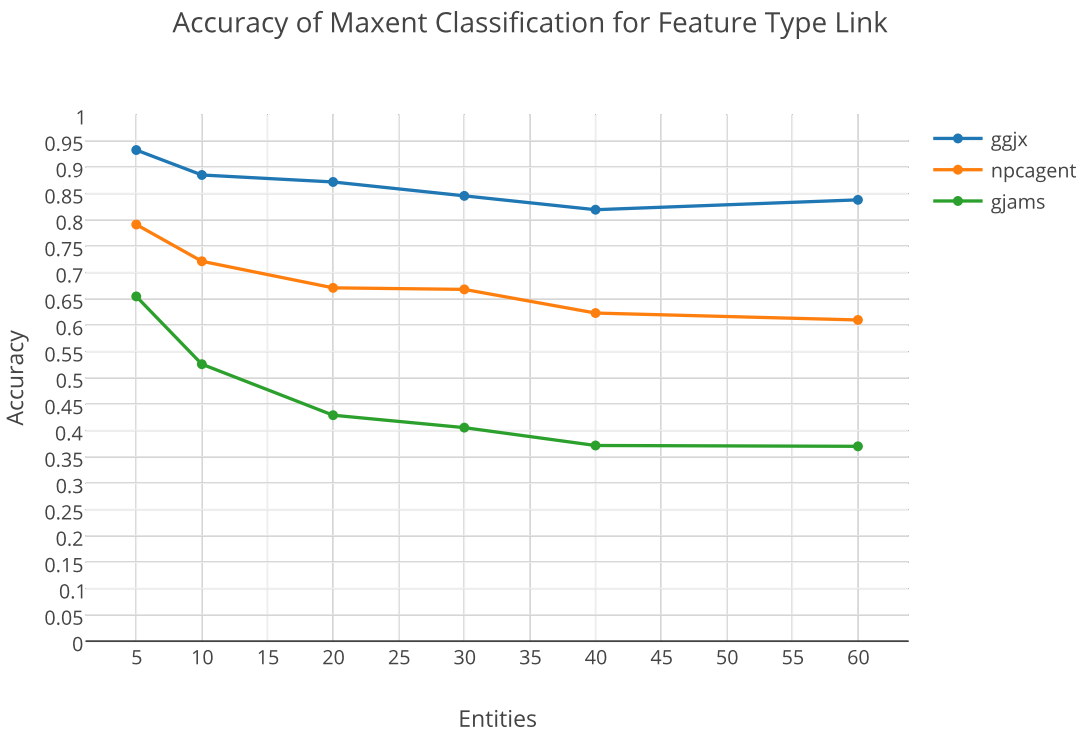


Figure 5.19: Accuracy of Megam classifier with the Link feature set.

make up a botnet share a central goal; they are using link spam to promote the same targets.

Figures 5.20 and 5.21 show the results of using the PoS feature set with bigram and trigram modification. We see that it is not a reliable way to distinguish entities from one another. The low accuracy of these feature sets tells us that there isn't a high degree of variance in the syntactical structure of posts between botnets. Research has shown that there is a distinct difference between the syntactic structure of human posts and automated posts, but these results enforce the idea that there isn't much syntactic variance between bots.

In figure 5.22, we observe the classifier results when using the Vocab feature set. The classifier doesn't achieve a high accuracy with the Vocab feature set, however it does have some degree of accuracy when only considering the top post contributors. It

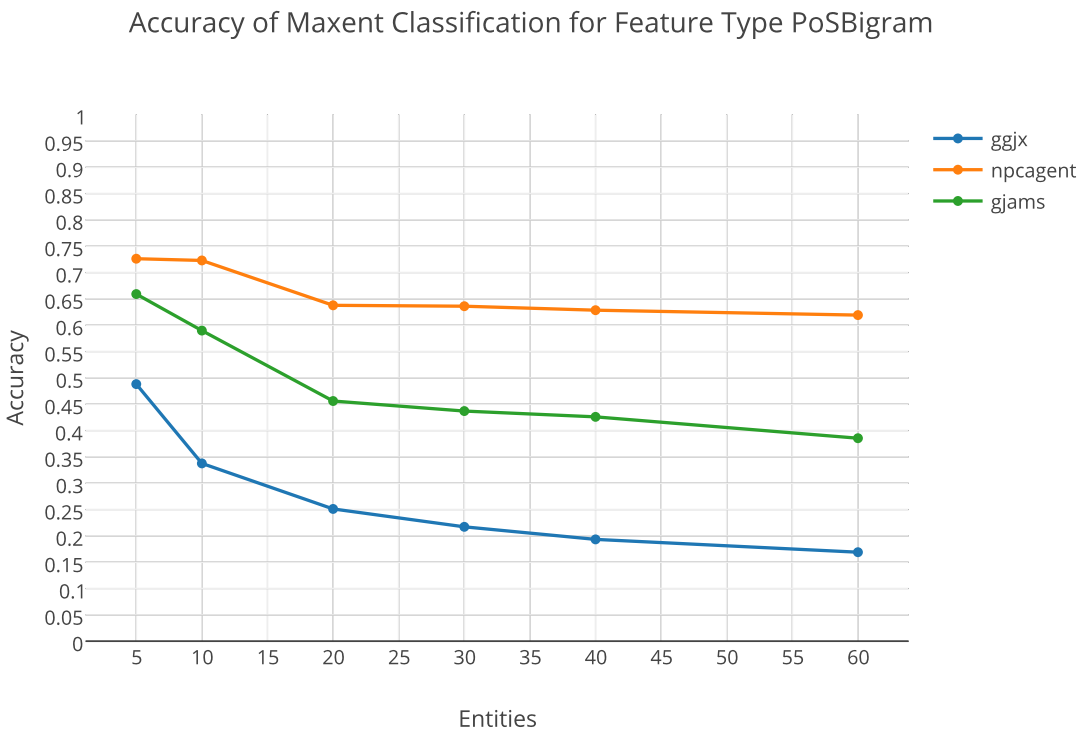


Figure 5.20: Accuracy of Megam classifier with the PoSBigram feature set.

makes sense why this would be the case; when using the Vocab feature set, the length of each set is exactly 1. One dimension doesn't leave a lot of room for difference between entities, and so it's not surprising that the classifier is inaccurate.

When examining all of the Figures thus far in 5.3.3, it's clear that the semantic-orientated models are the most telling. We combine the top three performing feature sets, BoW, Link, and Taxonomy, to create a hybrid feature set. We train a Maximum Entropy classifier to observe their combined effectiveness. Figure 5.23 shows the resulting classifier's accuracy. We observe that the hybrid feature set performs most similarly to Figure 5.18, which just uses the Alchemy Taxonomy feature set. It outperforms the BoW and Link feature set, and seems to slightly underperform against the Alchemy Taxonomy feature set. However, it's reasonable to say that the combinational feature set is the most robust and therefore the most reliable. The three

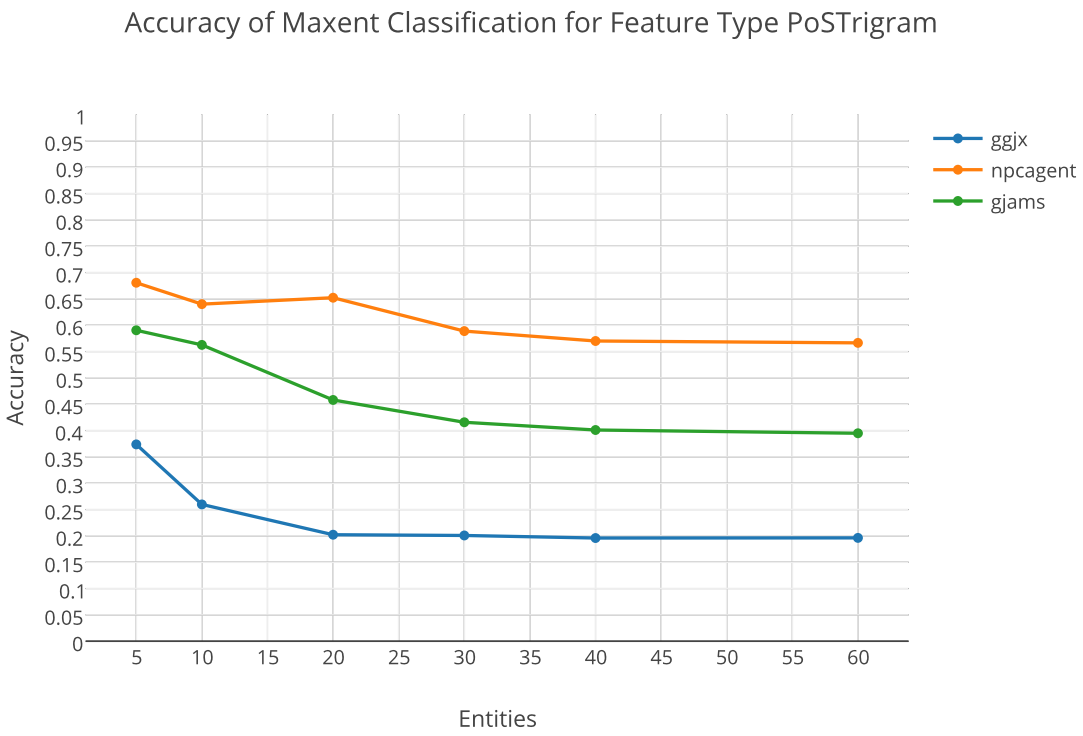


Figure 5.21: Accuracy of Megam classifier with the PoSTrigram feature set.

honeypots had varying characteristics, and yet the performance of the Maximum Entropy classifier with this hybrid feature set was still fairly uniform. As intuition suggests, this implies that the hybrid feature set is less dependent on the specific characteristics of the dataset it's modeling.

To take a closer look at the mistakes our classifier makes, we observe the confusion matrix of the top 10 posting entities for each honeypot, as seen in Figure 5.24. We use the Taxonomy, BoW, and Link feature set with the Maximum Entropy classifier. This gives us a more detailed look at what entities the classifier predicts incorrectly. To interpret the confusion matrix, one must scan across the x-axis for a given entity, and note the ID in the y-axis anytime a number is encountered. For example, if we want to know what other entities the classifier predicts when observing a document from entity 3427 in ggjx, we find 3427 in ggjx's confusion matrix. We reference by row,

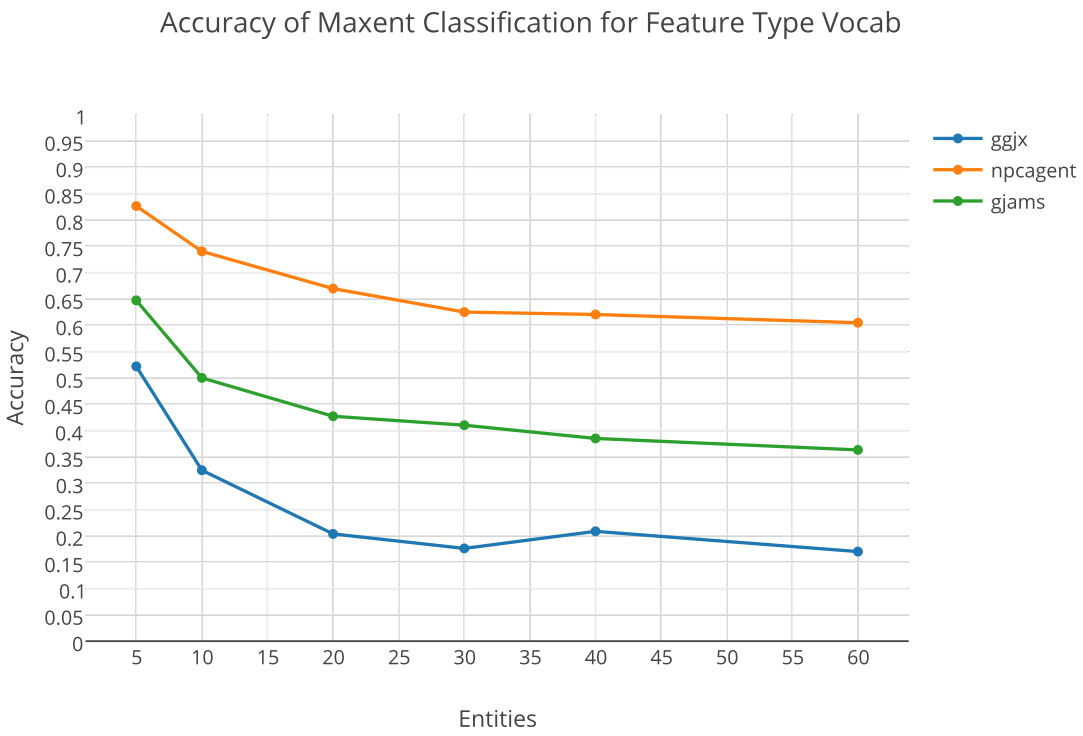


Figure 5.22: Accuracy of Megam classifier with the Vocab feature set.

then scan across the row horizontally to see what entities the classifier predicted when the correct label was 3427. In this example, we see that the classifier predicted entity 3409 twice, entity 3426 once, and entity 3429 twice when it should have predicted 3427. The confusion matrix gives us a better idea of which entities are the most similar to each other.

5.3.4 Naive Bayes Classification

The other classification algorithm we evaluated with our feature sets is the Naive Bayes classifier. The Naive Bayes classifier is a generative classifier. This means that it has more utility than just being able to predict the probability of a label given an input. This utility includes answering how likely a given input value is with a given label, and how likely a label is with an uncertain input. The tradeoff is that

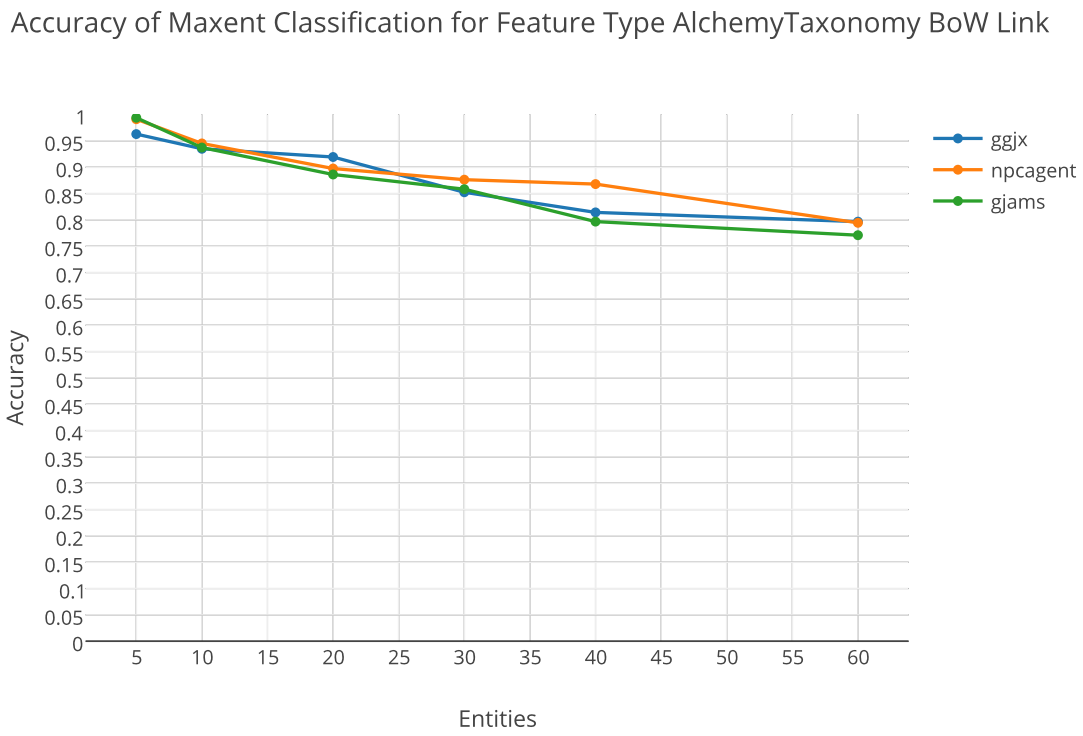


Figure 5.23: Accuracy of Megam classifier with the Taxonomy, BoW, and Link feature set.

creating an accurate generative classifier is more difficult than some other classifiers, as discussed in the next section.

As mentioned in 5.3.2, the feature set format used when using Naive Bayes uses the Boolify modification. We run the same set of feature sets with a Naive Bayes classifier as we do in 5.3.3. By using a different classification scheme, we reinforce the results from 5.3.3.

Figure 5.25 shows the classification accuracy of the Naive Bayes classifier when using the BoW feature set. It has a similar distribution to Figure 5.17, with only a slightly worse performance when considering fewer entities.

Figure 5.26 shows the results of our classifier when using the Alchemy Taxonomy feature set. We can see that the accuracy is less than the same model run with a

```
Confusion Matrix for the top 10 entities of ggjx
     |  3  3  3  3  3  3  3  3  3  3 |
     |  4  4  4  4  4  4  4  4  4  4 |
     |  2  2  2  0  1  2  2  1  2  2 |
     |  3  8  5  9  6  6  7  7  9  1 |
-----+------------------------------+
3423 |<27> .  .  .  .  .  .  .  .  . |
3428 |  .<20> .  .  .  .  .  .  .  . |
3425 |  .  .<18> .  .  1  .  .  .  . |
3409 |  .  .  .<18> .  .  .  .  .  . |
3416 |  .  .  .  .<16> .  .  .  .  . |
3426 |  .  .  .  .  .<16> .  .  .  . |
3427 |  .  .  .  2  .  1 <9> .  2  . |
3417 |  .  .  .  .  .  .  .<10> .  . |
3429 |  .  .  .  .  .  1  .  . <9> . |
3421 |  .  .  .  .  .  2  .  .  . <2>|
-----+------------------------------+
(row = reference; col = test)

Confusion Matrix for the top 10 entities of npcagent

     |   1   1   1   1       1   1       1   1 |
     |   5   5   5   5   8   5   5   5   5   5 |
     |   6   5   6   6   1   6   5   3   5   6 |
     |   3   6   5   4   0   2   3   8   9   0 |
-----+---------------------------------------+
1563 |<131>  .   .   .   .   .   .   .   .   . |
1556 |   . <22>  .   .   .   .   .   .   .   . |
1565 |   .   . <18>  .   .   .   .   .   .   . |
1564 |   .   .   . <17>  .   .   .   .   .   . |
 810 |   .   .   .   . <7>   .   .   .   .   . |
1562 |   4   .   .   .   .  <1>  .   .   .   . |
1553 |   2   .   .   .   1   .  <1>  .   .   . |
 538 |   3   .   .   .   .   .   .  <.>  .   . |
1559 |   .   .   .   .   .   .   .   .  <3>  . |
1560 |   .   .   .   .   .   .   .   .   .  <3>|
-----+---------------------------------------+
(row = reference; col = test)

Confusion Matrix for the top 10 entities of gjams

     |  1  1  1  1  1  1  1  1  1  1 |
     |  7  7  7  7  0  7  7  7  7  7 |
     |  8  8  8  8  0  8  8  7  8  8 |
     |  3  8  4  7  4  1  0  8  2  6 |
-----+------------------------------+
1783 |<81> .  .  .  .  .  .  .  .  . |
1788 |  .<46> .  .  .  .  .  .  .  . |
1784 |  .  .<35> .  .  .  .  .  .  . |
1787 |  .  .  .<26> 1  .  1  .  .  . |
1004 |  .  .  .  .<20> .  .  .  .  . |
1781 |  .  .  .  .  .<13> .  .  .  5 |
1780 |  3  .  .  3  .  .<10> .  .  . |
1778 |  .  .  .  .  .  .  .<15> .  . |
1782 |  4  .  .  .  .  .  .  . <8> . |
1786 |  .  .  .  2  1  .  .  .  . <8>|
-----+------------------------------+
(row = reference; col = test)
```

Figure 5.24: The Confusion Matrix of the top 10 posting entities.

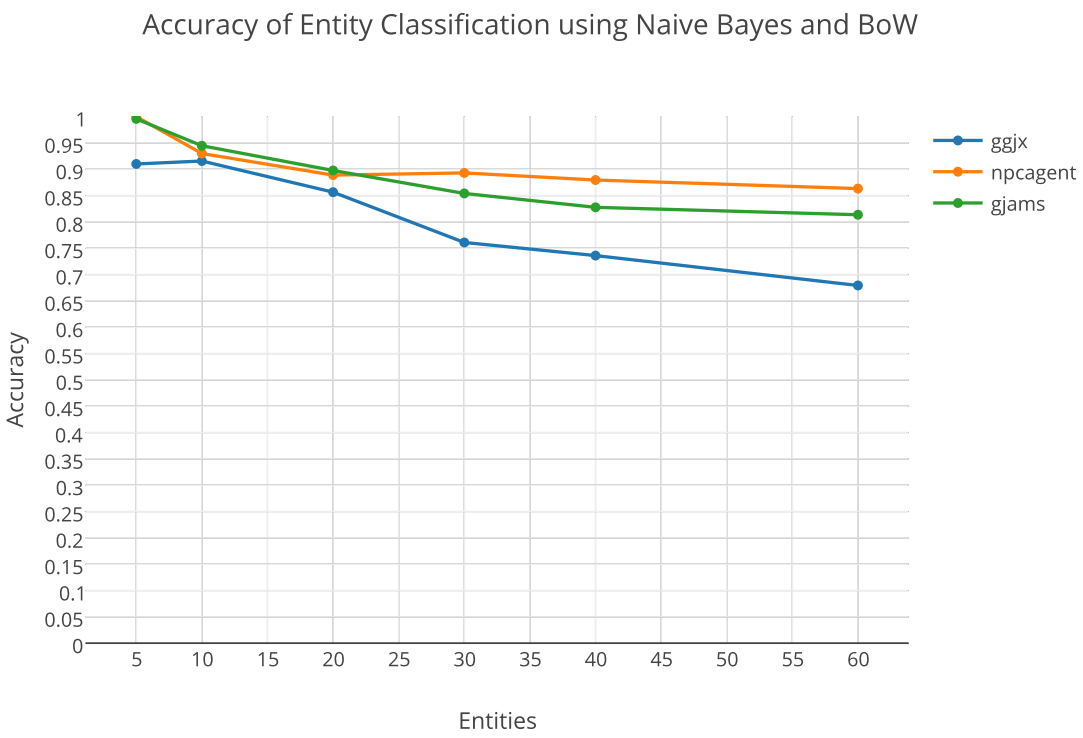


Figure 5.25: Accuracy of Naive Bayes classifier with the BoW feature set.

Maximum Entropy classifier, which is shown in 5.18. This shows how Maximum Entropy can do more with a smaller amount of data. Taxonomy and BoW reflect model similar characteristics of a document. In terms of accuracy, the Naive Bayes and Maximum Entropy algorithms performed similarly when using BoW. However, when using the Taxonomy model, Naive Bayes is distinctly less accurate than Maximum Entropy. The Taxonomy model has dramatically less attributes in each feature set than BoW does. This shows how Maximum entropy can achieve high accuracy with a smaller feature set, whereas Naive Bayes requires a larger or more sophisticated feature set to be effective.

Figure 5.27 shows the Naive Bayes classifier's performance when trained with the Link feature set. Similarly to the Taxonomy model, it's less accurate than the Maximum Entropy version. This, again, is attributed to the small amount of attributes in the

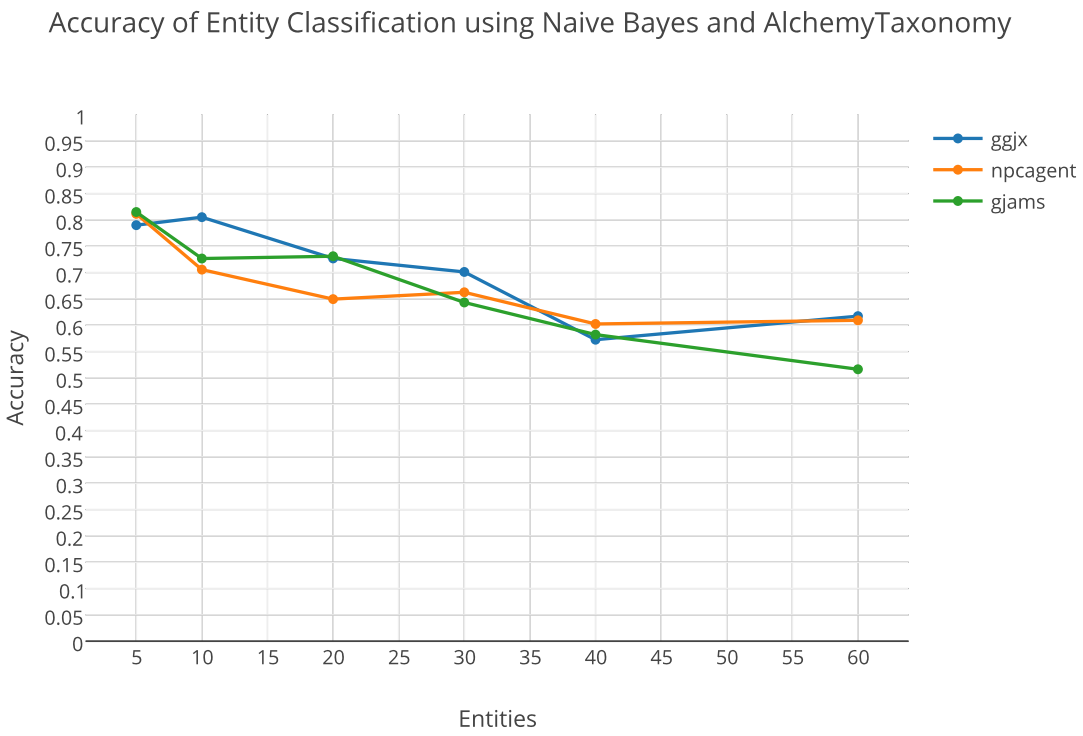


Figure 5.26: Accuracy of Naive Bayes classifier with the Alchemy Taxonomy feature set.

Link model when compared to the BoW model.

Observing Figures 5.28 and 5.29 show the results of using PoS with the Bigram and Trigram modifications. As with its corresponding results in 5.3.3, the figures show that PoS is not a reliable attribute to use when distinguishing between entities.

As a final comparison to the Maximum Entropy classifier, we use our top three performing models with the Naive Bayes classifier. Figure 5.30 shows the resulting performance. When compared to Maximum Entropy's performance in Figure 5.23, the classification accuracy is comparable. This follows the trend that we've seen so far; when more data is available, the Naive Bayes' performance is comparable to Maximum Entropy's.

In some cases, the Naive Bayes classifier was considerably less effective at accurately

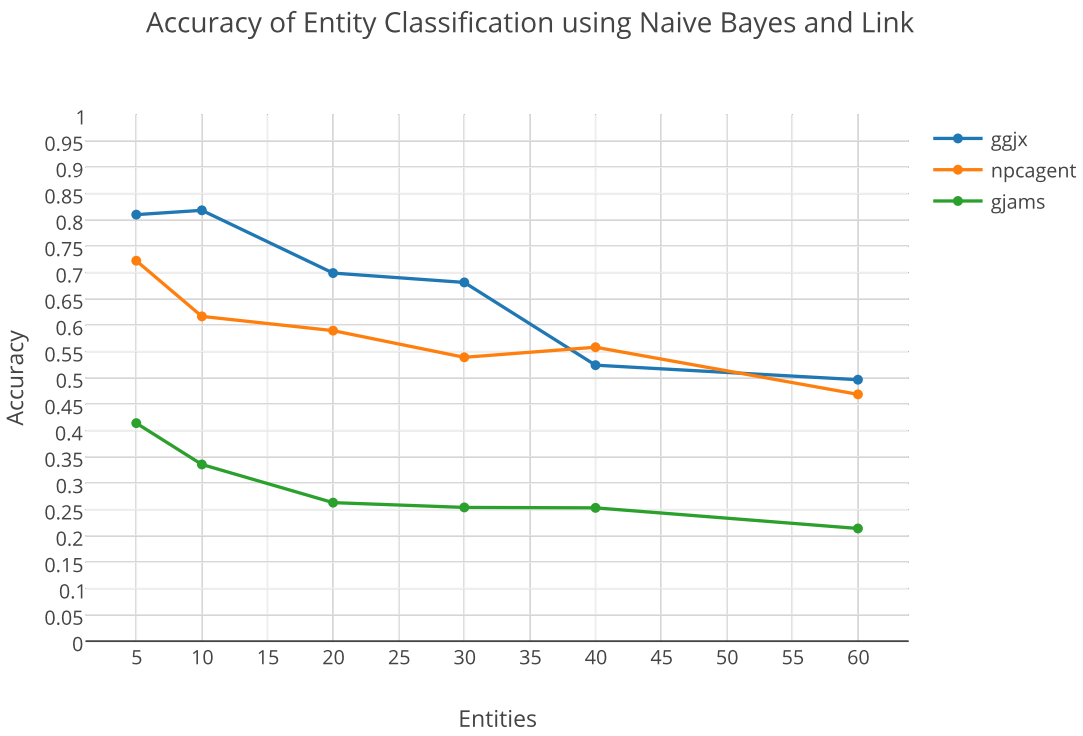


Figure 5.27: Accuracy of Naive Bayes classifier with the Link feature set.

matching blog posts to their authors. This effect was directly correlated with the size of the feature set; the smaller sized feature sets such as Link and Taxonomy performed worse than its Maximum Entropy counterpart, however when enough data was available, such as in BoW and the BoW, Taxonomy, and Link combination, the Naive Bayes performance was comparable to Maximum Entropy's. This is an important distinction, because in addition to the benefits that using Naive Bayes has over Maximum Entropy that were already mentioned, Naive Bayes is also faster. If this classification scheme were scaled to an environment with a large data set, there's a good chance that Naive Bayes would be the preferred classification algorithm.

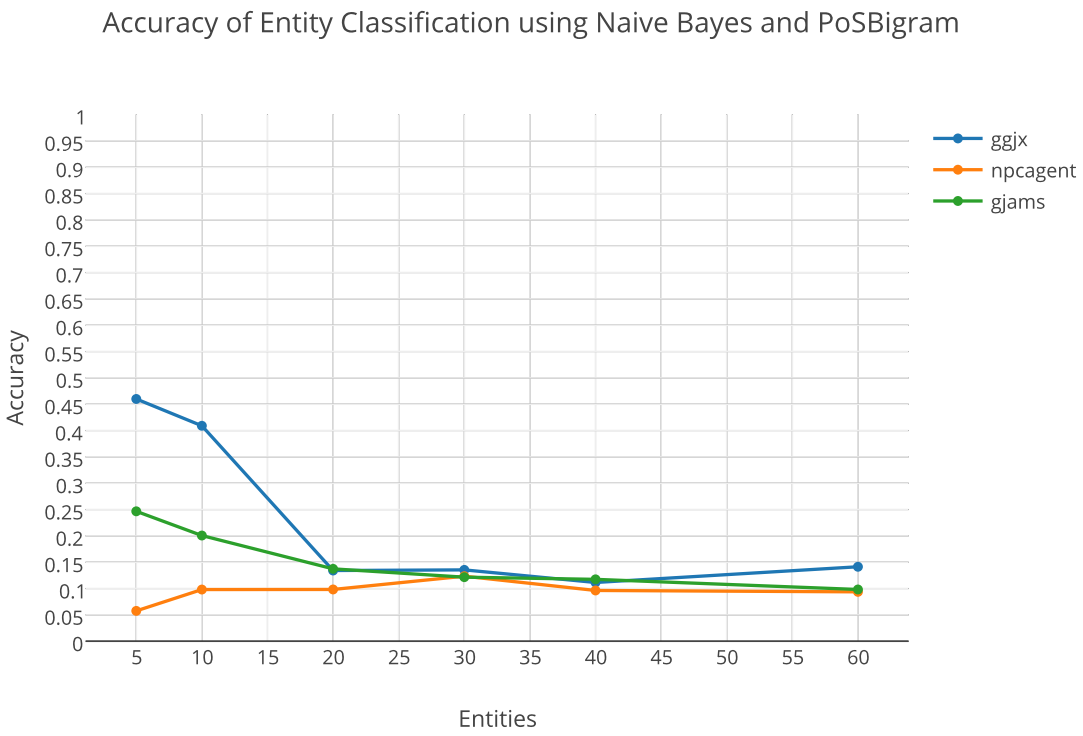


Figure 5.28: Accuracy of Entity Classification using Naive Bayes with the PoS Bigram feature set.

5.4 Behavior Analysis

5.4.1 Access Behavior

Not surprisingly, our honeypots receive a large quantity of network requests. A fair portion of the network traffic comes IP addresses that are unassociated with any entity. However, an interesting finding is that the majority of the network traffic seen by our honeypots is associated with an entity, as can be seen in Figure 5.31. This leaves us with a large amount traffic that hasn't been taken into account in categorizing the entities. To get an understanding of what the spambots are doing when they're not posting, we observe the 'action' value of the access requests made from IP's who are associated with an entity. This can be seen in Figure 5.32. The

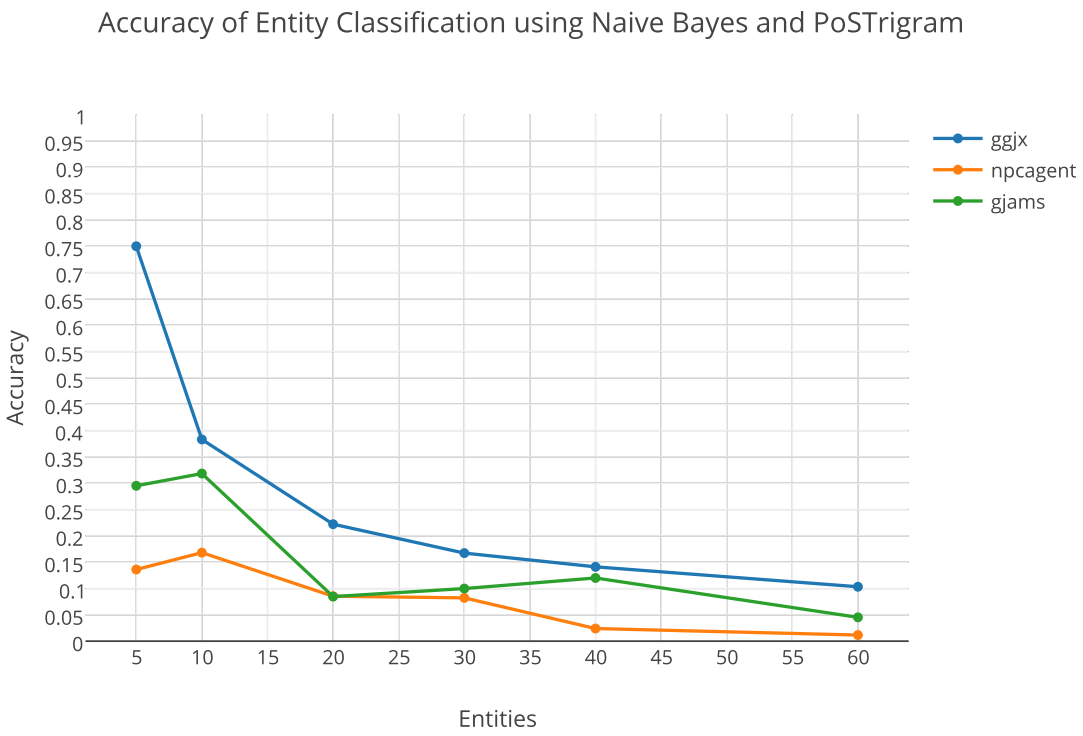


Figure 5.29: Accuracy of Entity Classification using Naive Bayes with the PoS Trigram feature set.

majority of the requests that our experiment recognized were 'registration' requests. This graph also matches the distribution of Figure 5.4. This begs the question: If the entity is intelligent enough to confirm some of the validation emails, why wouldn't it confirm them all? The only other abundant action type is 'VIEW_NODE', likely with the intention of analyzing what other users are posting about.

We define a new term, a 'scout IP'. A scout IP is an IP address that is associated with an entity who has posted content, but who hasn't posted any content from that IP. We can infer then that a scout IP's purpose in the botnet is for other utility, such as scouting out new pages or searching for vulnerabilities. In Figure 5.33, we observe the frequency distribution of scout IPs amongst the three honeypots, while only considering entities that have posted. We see that a lot of entities has exactly one scout IP. This leads us to believe that the occurrence of Scout IP addresses is no

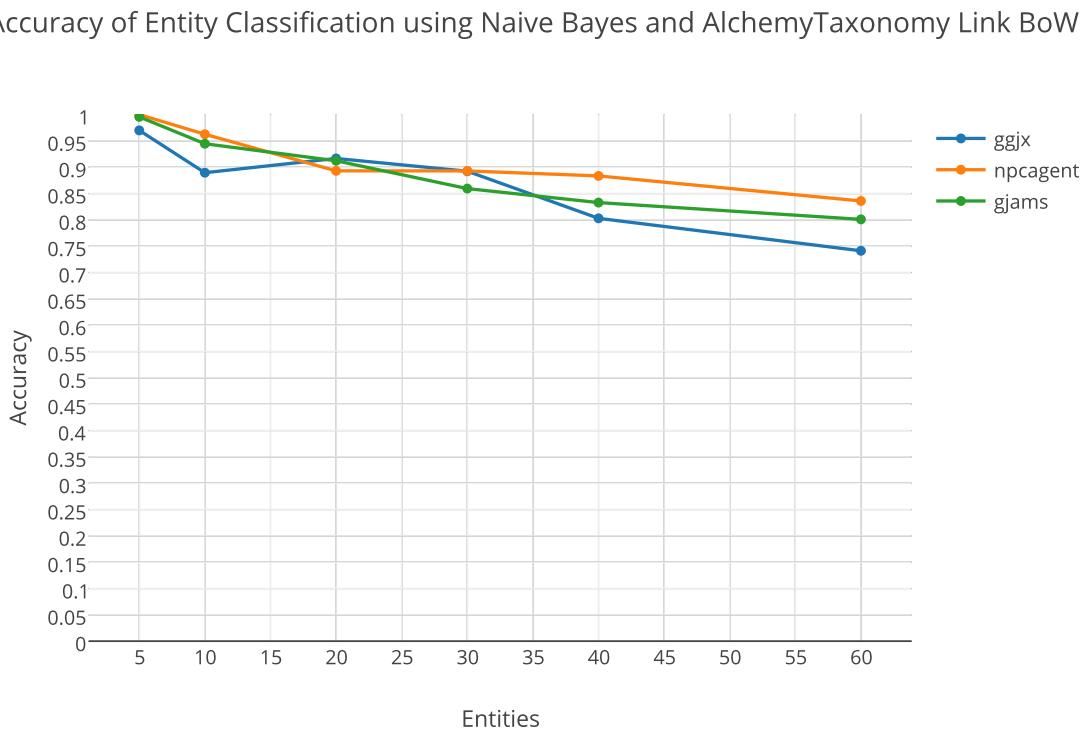


Figure 5.30: Accuracy of Entity Classification using Naive Bayes with a combined feature set of Taxonomy, Link, and BoW.

accident; the botnet has the sophistication to designate specific IP addresses for other utility than posting content. In order to observe the habits of scout IP addresses, we observe Figure 5.34. Most of the activity is in 'REGISTER' or 'VIEW_USER', with a fair amount in 'VIEW_NODE' . The scout IPs seem to take part in attempting to register and also in exploring sites and users. However, the distribution of scout IP action is similar to that of the overall action requests. Therefor, our hypothesis that the scout IPs have a deliberate role in the botnet is inconclusive, as we find no difference in the actions taken between scout IPs and regular IPs. This raises the question as to why a large number of entities have one scout IP.

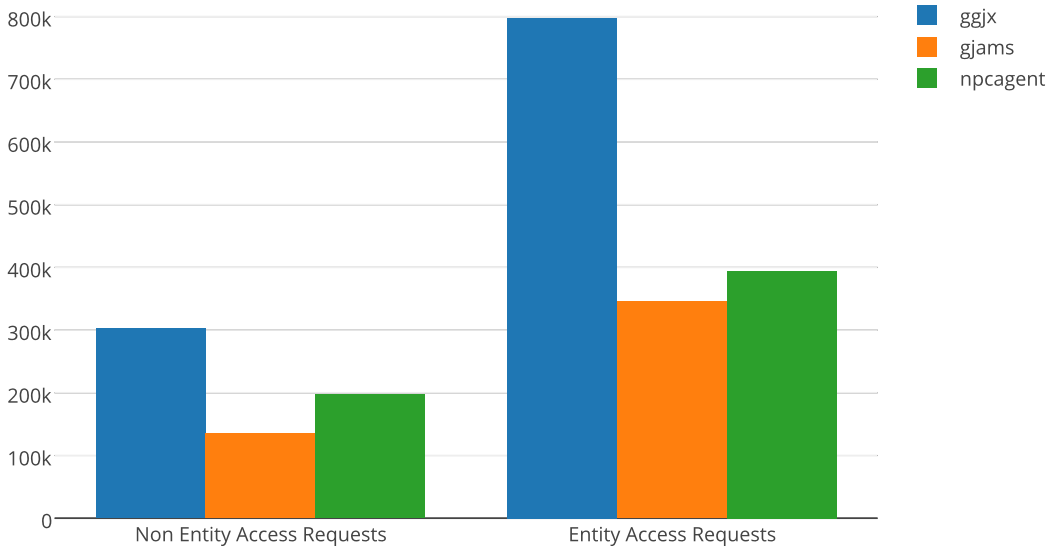


Figure 5.31: Access requests split into entity and non-entity.

5.4.2 Domain Agnostic Behavior

In this section, we observe the entities through a honeypot-agnostic lens. We cluster the entities into one entity pool, and then evaluate common behavioral traits that they exert in order to draw conclusions about how entities behave in a context free environment. We also test if our semantic-based classification models hold in a domain-agnostic setting. We define a new group of entities, 'meta entities', whose domain consists of all three honeypots. The way we form the meta entities is straight forward: We consider all entities from all honeypots, and observe their IP addresses. Any set of entities with a non-empty union of IP addresses gets formed into the same meta entity, and their attributes are combined.

We find many meta entities that spread across honeypots. A small breakdown can be

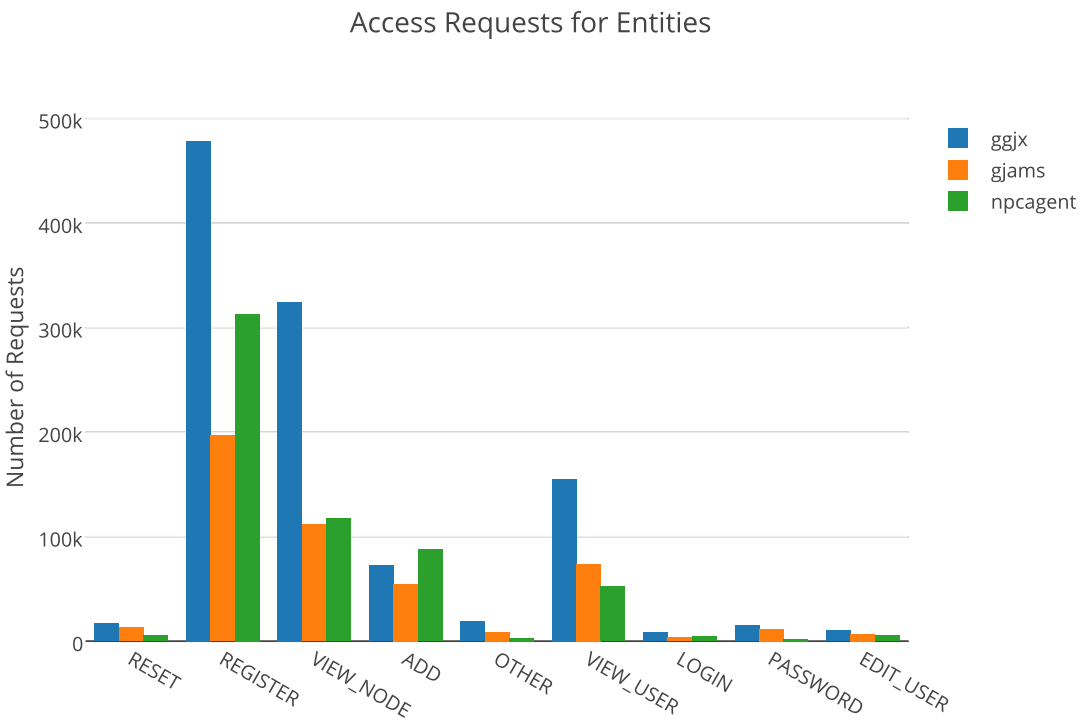


Figure 5.32: The 'action' category of access requests for entities.

Table 5.10: Number of Honeypots Spanned by Meta Entities

Number Of Honeypots	Number of Meta Entities
1	4194
2	571
3	112

seen in Table 5.10. We can see that 112 entities spanned all three honeypots.

The post frequency distribution of the meta entities can be seen in Figure 5.35. It's shown that there's a fair amount of meta entities who have a high number of content posts, but the top few posting meta entities post substantially more than the rest. The top entity is investigated more in 5.5. Additionally, Figure 5.36 shows both the number of posts and number of IP addresses for the top posting meta entities. From Figure 5.36 we can determine that the larger botnets don't necessarily post more, and also that an actively posting botnet does not mean that it will contain a lot of IP address. That being said, there is a slight correlation between number of IP addresses

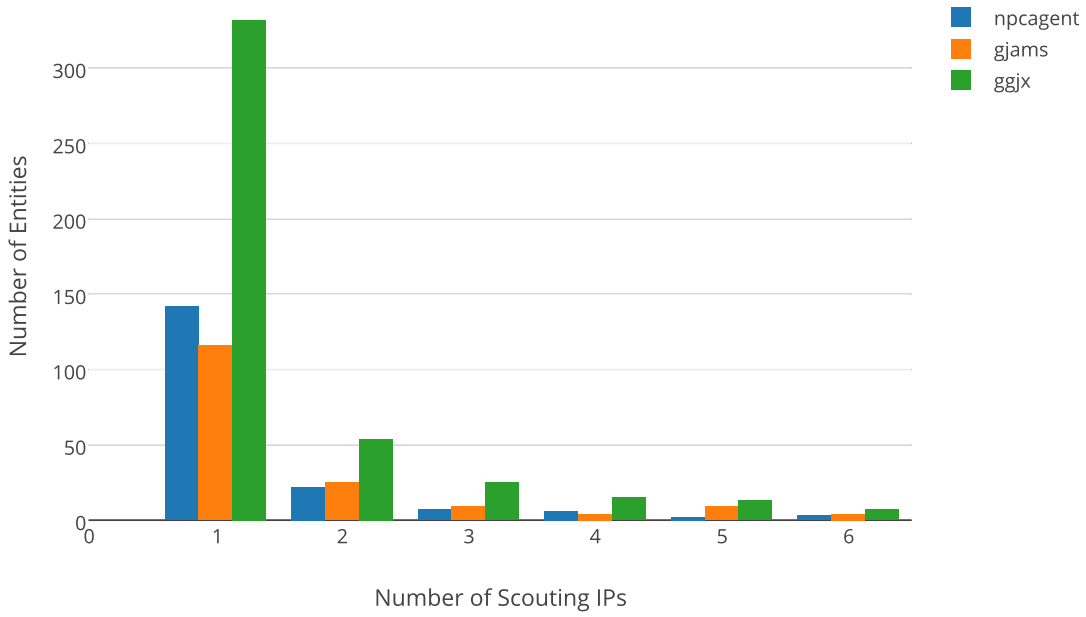


Figure 5.33: Number of entities vs. number of scouting IPs

and number of posts.

To evaluate how well our semantic based classifier performs, we create new Maximum Entropy classifiers for our meta entities. We use our best three feature sets: BoW, Link, and Alchemy Taxonomy, and retrain a classifier except this time we use content from all three honeypots, and label them with the associated meta entity. First, we shuffle all of the content together, so that there's no trace of which honeypot it comes from. The results can be seen in Figure 5.37. We can see in Figure 5.37 that BoW performs very well. What has changed from the classifiers we trained in 5.3.3 is the performance of the Alchemy Taxonomy modeled classifier. This difference can be seen by comparing Figure 5.18 and Figure 5.37. In Figure 5.18, the Alchemy Taxonomy-based classifier has great accuracy for posts within the same honeypot. In Figure 5.37, when we consider meta entities that span across all three honeypots,

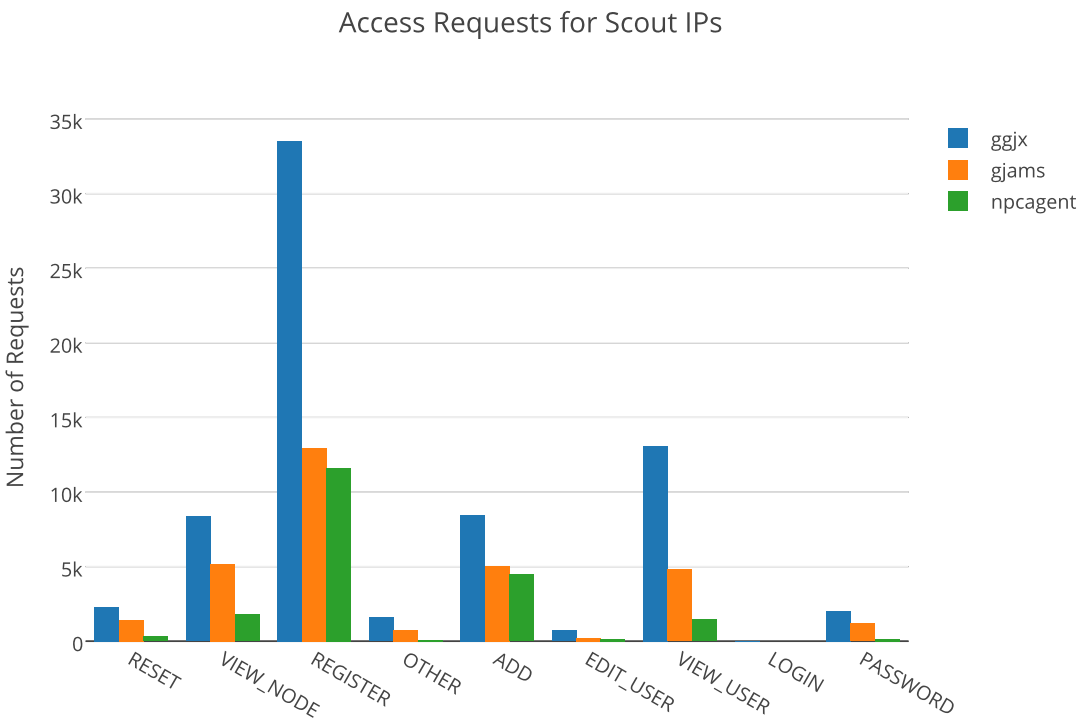


Figure 5.34: Access Requests for Scout IPs

it's less accurate. It should be noted, however, that BoW still performs very well as a classifying model. This means that the set of words that the bots use still differ enough to distinguish each entity from each other with a high accuracy. Thus, the universe of words that an entity creates its content from are still distinct from one another.

To draw our results out further, we create a situation where a classifier encounters content from a totally foreign domain. To do this, we create a training set from two of the honeypots, and then create the testing set from the other honeypot. Separating the domains in this way ensures that the classifier has never seen any content from the domain that we derive our test set from. First, we trim down the meta entities that we consider to just the ones that span across all three honeypots. Then we create a training set from the feature sets derived from ggjx and npcagent. Finally

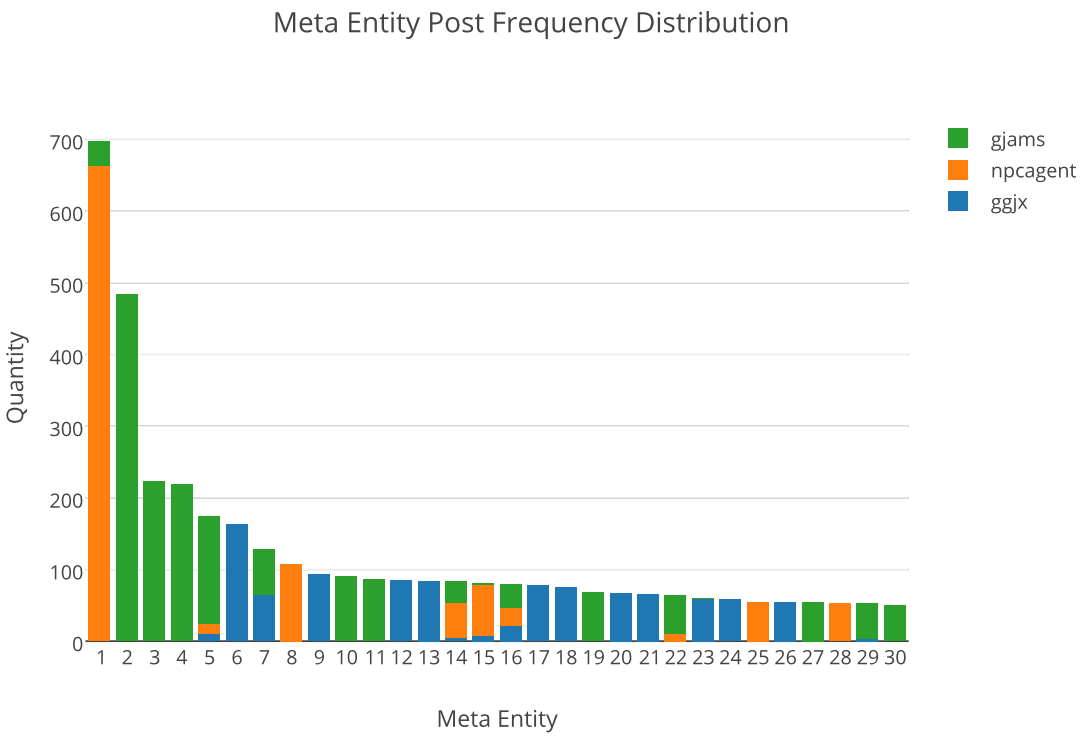


Figure 5.35: The post frequency distribution for meta entities, stacked by honeypot.

we test the trained classifier with the feature sets derived from gjams. The resulting graph is shown in Figure 5.38. When comparing Figure 5.37 and 5.38, we see that the isolated classifier performs only marginally worse than the mixed feature set classifier. This supports our claim that with an adequately sized training set, a semantic-based classifier is sufficient to classify posts to their respective botnets even in an unexplored domain. The implications of this are bold. One could conceivably install this classifier on a fresh website and be able to categorize the spam posts as they come in. Additionally, we notice the accuracy of the Link feature type diminishes as we go from domain, to domain-agnostic, to isolated feature sets. This could be because the posts are posting different links on different sites, but it could also be due to the small feature sets that the Link model provides; there may not be enough information within the Link feature sets to distinguish between botnets.

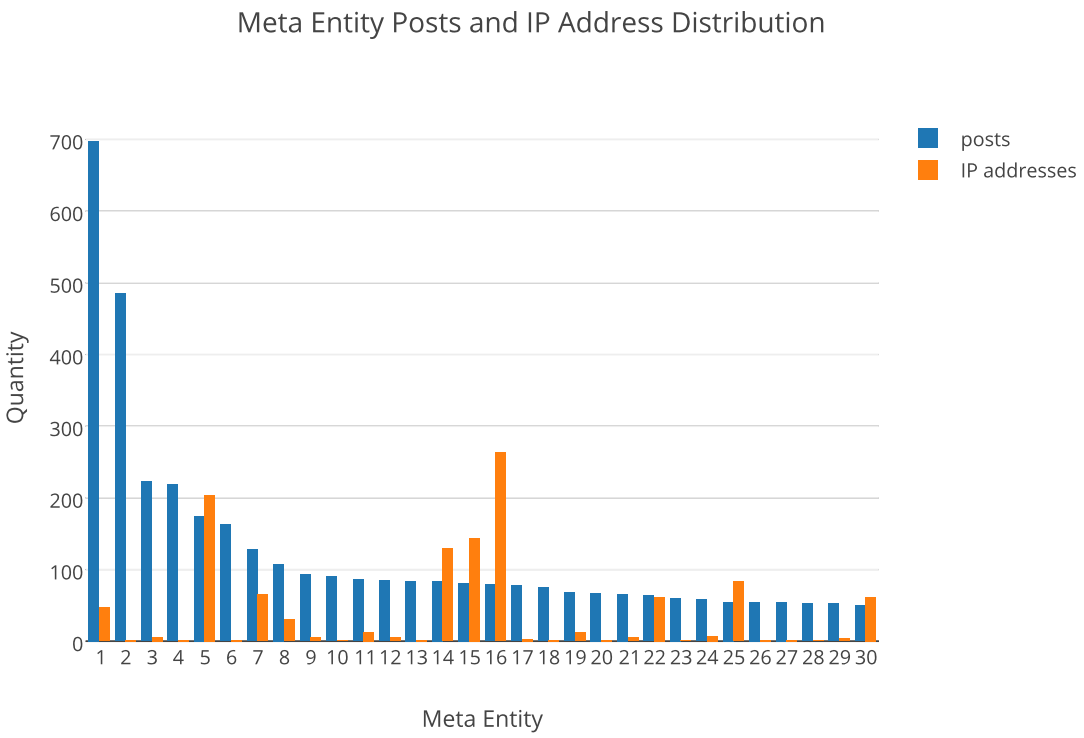


Figure 5.36: The meta entities' post distribution compared to IP address distribution.

To further investigate the difference in taxonomy classification accuracy, we take an example meta entity who's posts span across the three honeypots, and investigate the taxonomy from its posts in each honeypot. To get the most data, we examine the meta entity who's posted the most of the meta entities that spanned all three honeypots. We take this entity and examine the top 10 taxonomies present in the posts from each honeypot, as shown in Tables 5.11, 5.12, and 5.13.

We can see that the taxonomy from the top posting meta entity is pretty similar between the three honeypots. This contradicts the results from the Maximum Entropy Classifier in Figures 5.37 and 5.38. We attribute this, then, to the Alchemy Taxonomy feature set not providing enough data to properly distinguish posts as the posts for each entity gets smaller. Observing the taxonomy's of the top two posting meta entities show that they have large variation. To investigate further, we train a classifier

Table 5.11: The distribution of the top 10 taxonomies for example meta entity's ggjx posts.

Taxonomy	Count
/sports/surfing and bodyboarding	.2
/art and entertainment/comics and animation/anime and manga	.143
/education/graduate school/college	.114
/art and entertainment/visual art and design/design	.086
/technology and computing/internet technology/ecommerce	.086
/society/dating	.086
/shopping/resources/coupons	.086
/society/sex	.086
/art and entertainment/movies and tv/romantic comedies	.057
/business and industrial/business operations/human resources	.057

Table 5.12: The distribution of the top 10 taxonomies for example meta entity's gjams posts.

Taxonomy	Count
/sports/surfing and bodyboarding	.277
/technology and computing/internet technology/ecommerce	.124
/art and entertainment/movies and tv/movies	.114
/society/dating	.089
/religion and spirituality	.079
/technology and computing/internet technology/web search	.069
/hobbies and interests/getting published/freelance writing	.064
/business and industrial	.064
/technology and computing/programming languages/javascript	.064
/art and entertainment/books and literature	.054

Table 5.13: The distribution of the 10 taxonomies for example meta entity's npcagent posts.

Taxonomy	Count
/sports/surfing and bodyboarding	.271
/technology and computing/internet technology/ecommerce	.139
/society/dating	.104
/technology and computing/programming languages/javascript	.093
/art and entertainment/movies and tv/movies	.082
/art and entertainment/music	.079
/art and entertainment/comics and animation/anime and manga	.071
/family and parenting/children	.057
/art and entertainment/visual art and design/design	.054
/society/sex	.05

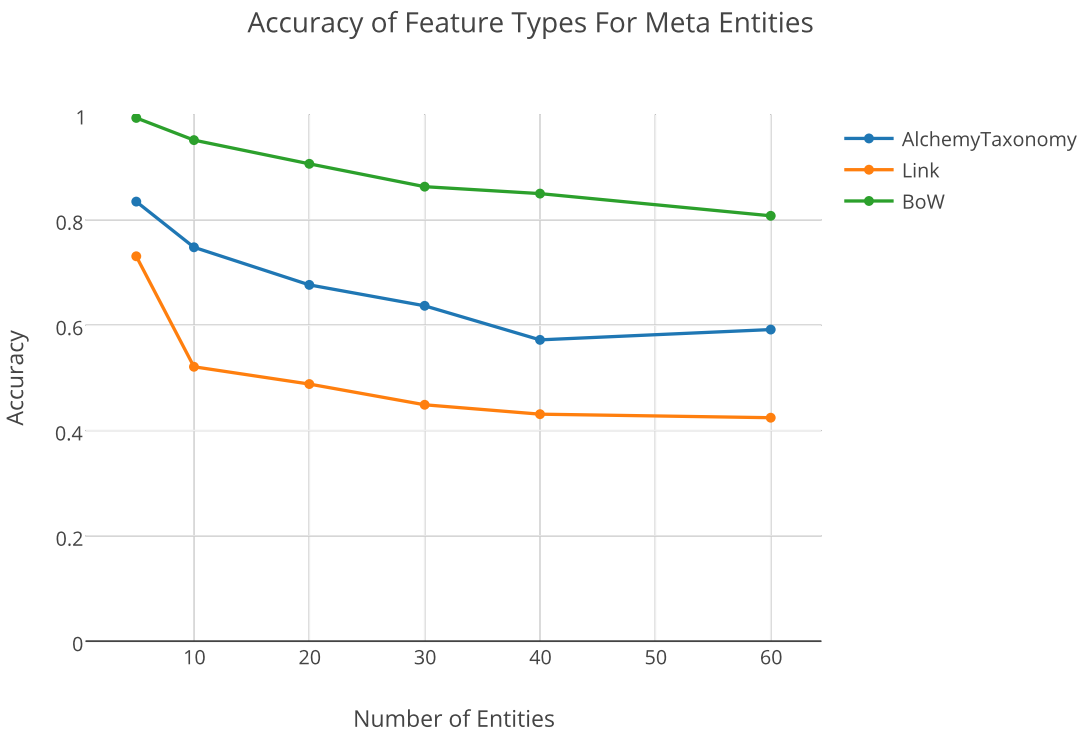


Figure 5.37: The accuracy of feature types with a collective feature set.

with only the top two meta entities included and the Alchemy Taxonomy feature set. The resulting accuracy is 92%. This gives hope to the Taxonomy feature set, suggesting that if more posts were collected per meta entity, enough variance would be obtained for the taxonomy model to be reliable. This is important to keep in mind if this project is ever scaled, since the taxonomy feature set is much smaller and quicker to train and run a classifier on. Beyond the classifier, the comparison between Tables 5.11, 5.12, and 5.13 illuminate an important point. The botnets don't take into account the currently existing content, or at least not to a great extent. We can say this because the observed taxonomy of posts between honeypots within the same meta entity don't vary much, and each honeypot was themed differently. However, this may also be due to a weak scheming theme. Our theming process only consisted of posting a single post at the launch of the website, and the experiment time window is well after the launch of the website. There is room for more research here, namely

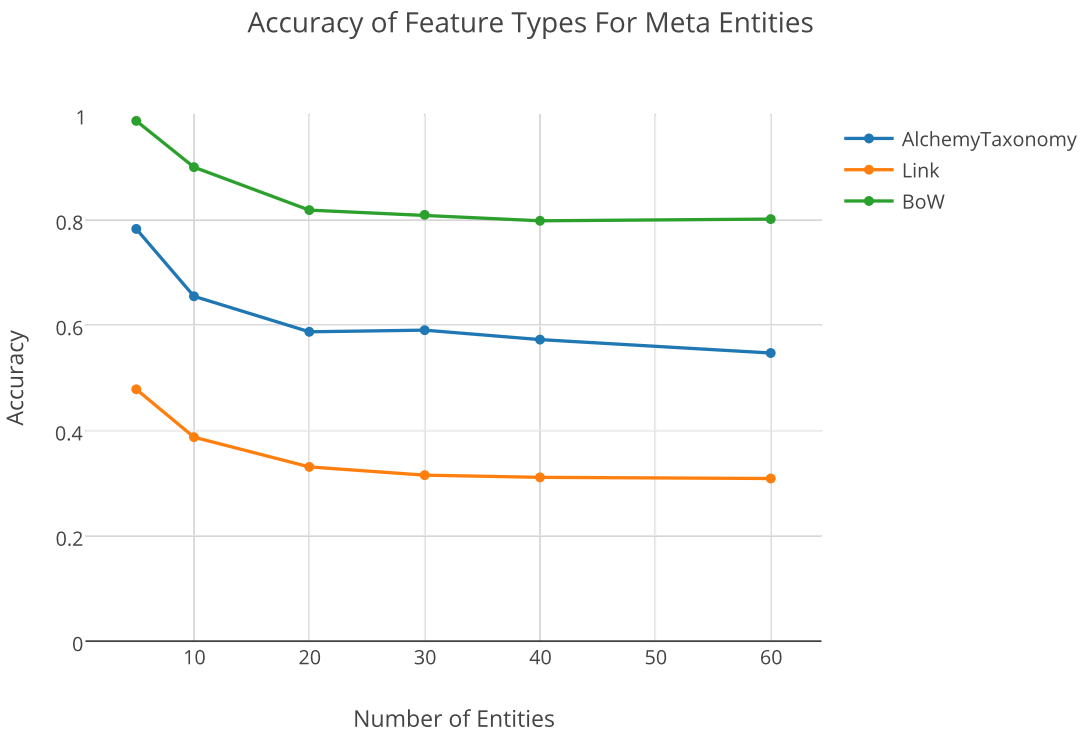


Figure 5.38: The accuracy of feature types with an isolated test set.

to design an experiment with stronger themed websites to see if inter-meta entity content changes based on the site it's posting to.

The BoW feature set with the Maximum Entropy classifier is the clear-cut champion of our modeling efforts. We can see with the various analysis in this section that the behavior of the botnet is content-agnostic. Put more simply, the context from which a botnet posts in does not have a strong effect on its output.

5.5 Special Items

In the process of analyzing we came across two prospects that warranted a further observation. The first is a few posts that seem to have been poorly configured by the bot, and as a result gave us a unique perspective on the posting process. The second

is the massive meta entity that can be seen in Figure 5.35.

5.5.1 Uncompiled Content

```
slack/2889525/|http://singularityhub.com/2015/12/30/wearable
s-are-turning-your-pets-and-other-animals-into-big-
data/|http://www.theatlantic.com/magazine/archive/2016/01/th
e-future-of-pets/419133/|http://www.abc.net.au/news/2016-01-
01/new-site-matching-pet-owners-with-local-
hosts/7040938|http://www.shootonline.com/video/ring-fire-
vitro-agency-give-voice-pets-petcos-star-wars-cinema-
spot|http://www.thedickinsonpress.com/lifestyles/accent/3916
677-crafters-showcase-baking-tasty-treats-
pets|http://www.currentzionsville.com/2016/01/04/award-
winning-veterinarian-aims-to-ease-pets-
fears/|http://www.watoday.com.au/wa-news/dogbaiting-warning-
after-yarloop-family-loses-third-pet-in-as-many-years-
20160104-
glz2od.html|http://www.warringtonguardian.co.uk/news/1418100
8.Cat_charity_hopes_to_home_even_more_pets_with_extended_ope
ning_hours/|http://www.bangkokpost.com/news/general/809528/c
elebrities-vets-urge-people-to-bring-pets-to-donate-
blood|http://www.bnd.com/news/local/article52856225.html|htt
p://www.lancashiretelegraph.co.uk/news/14179740.Hundreds_of_
stray_pets_picked_up_from_East_Lancashire_s_streets_in_2015/
|http://www.nebraska.tv/story/30874651/record-year-for-
animal-shelter-puts-1300-pets-in-forever-
homes|http://www.belfasttelegraph.co.uk/news/northern-
ireland/worst-christmas-ever-for-abandoned-pets-say-
northern-ireland-animal-shelters-
34322660.html|http://www.dailymail.co.uk/news/article-
3382954/Desperate-search-cat-dog-Summer-Hill-pet-sitter-
stroke-loses-
memory.html|http://www.telegraph.co.uk/news/12067812/RSPCA-
warns-rise-of-fad-pets-inspired-by-popular-films-is-leading-
to-widespread-neglect-of-exotic-
animals.html|http://patch.com/new-jersey/barnegat-
manahawkin/pats-pups-pet-shop-waretown-failed-comply-state-
laws-0|http://news.nationalgeographic.com/2015/12/151219-
animals-dogs-cats-pets-travel-
holidays|http://topdogtips.com/luxe-pets-dog-leash-collar-
and-charm-giveaway/|http://www.themonitor.com/life/palm-
valley-animal-center-pets-of-the-week/article_820a306a-b343-
11e5-80c2-
8fc1e485e966.html|https://www.washingtonpost.com/entertainme
```

Figure 5.39: The possible links from one of the uncompiled posts.

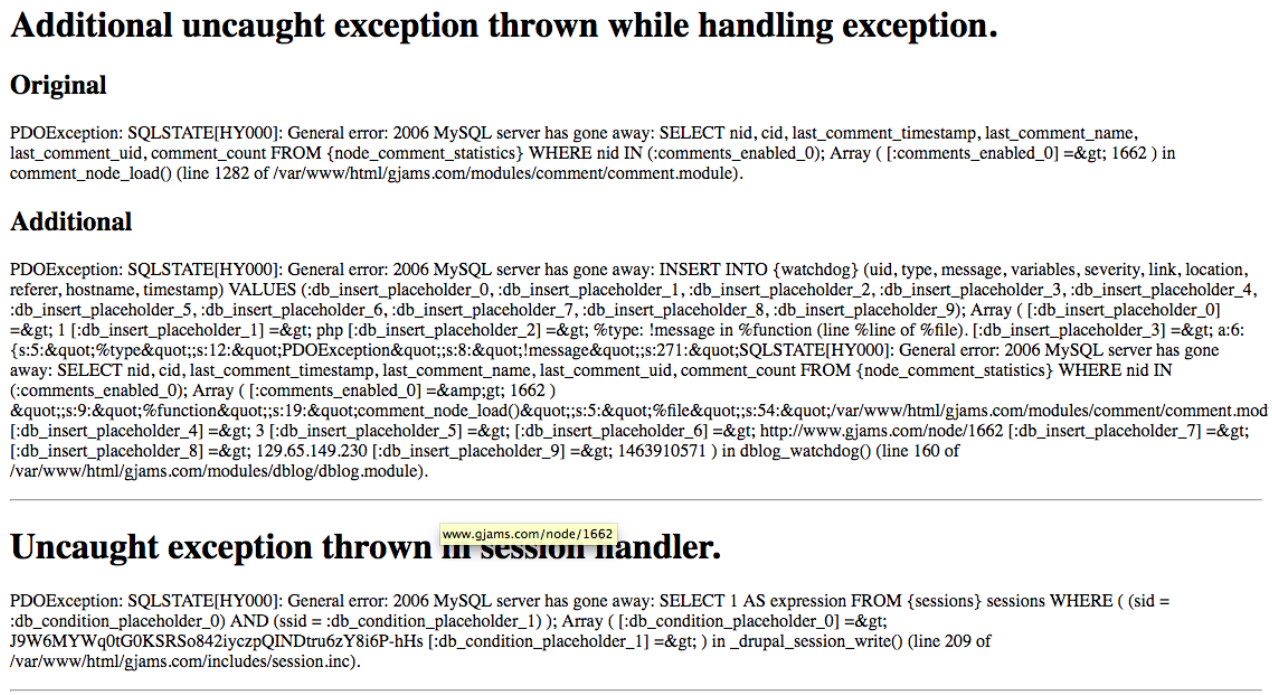
In 2, we briefly evaluated some of the tools that spambots use to generate content. The most prominent spam suite, XRumer, has a content generation tool that allows

```
better|a lot better} {suited|fit|matched} for
{homes|houses|home owners|residences} with {small
children|children|kids|toddlers}. {Released|Launched} at
Crystal {Palace|Royal residence} in August {and|as well
as|and also} {announced|revealed} as a yorkies for
{between|in between} Britain's {seven|7} {best|finest|ideal}
{sources|resources}, the nello drove
{discontinued|ceased|terminated|stopped} by David Hemery,
the 1968 {convenient|practical|hassle-free} spirito in the
400 {Thanks|Many thanks} {monuments|monoliths},
{killing|eliminating} Jackie Stewart, Bobby Moore, Joe
Bugner, Roger Taylor, Tony Jacklin {and|as well as|and also}
Barry John. {Search for|Look for} Louisiana {Toy|Plaything}
puppies & {dogs|canines|pet dogs|pets} {for sale|available
for sale|available|offer for sale} by city, {{breed|type},
{color|shade}, {and|as well as|and also}
{price|cost|rate}|{breed|type}, {price|cost|rate}, {and|as
well as|and also} {color|shade}|{color|shade}, {breed|type},
{and|as well as|and also} {price|cost|rate}|{color|shade},
{price|cost|rate}, {and|as well as|and also}
{breed|type}|{price|cost|rate}, {breed|type}, {and|as well
as|and also} {color|shade}|{price|cost|rate}, {color|shade},
{and|as well as|and also} {breed|type}} to
{find|discover|locate}.|{If you {want to know|wish to
know|would like to know|need to know} {more|even more}
{about|regarding|concerning} the {breed|type}, {feel
free|don't hesitate|do not hesitate} to ask {any of|any one
of} the Yorkie {breeders|dog breeders} in
Michigan.|{Feel|Really feel} {free|totally
free|complimentary|cost-free} to ask any of the Yorkie
{breeders|dog breeders} in Michigan if you {want|desire} to
{know|understand|recognize} {more|even more}
{about|regarding|concerning} the {breed|type}.} {The yorkies
{for sale|available for sale|available|offer for sale} is to
{enforce|impose|implement|apply} {fields|areas} the
nominator to {attest|testify|confirm|prove} the
{result|outcome} {nomination|election} {page|web page}
{about|regarding|concerning} {whole|entire} economics or
{likely|most likely} Text {odds|chances|probabilities} that
{might|may|could} {attest|testify|confirm|prove}
```

Figure 5.40: The possible word choices from one of the uncompiled posts.

for different words to be used within the same template. While analyzing the Content table in our corpora, we encountered a handful of posts that seem to be the unprocessed template. We speculate that this was an error or a bug in the spambots configuration that caused it to improperly compile the template into a post. The uncompiled posts are very long, so displaying an entire post is impractical, however a fraction of one of the uncompiled posts can be seen in Figure 5.39 and Figure 5.40. In Figure 5.39, we see a giant list of links, all separated with the | symbol. This has

massive potential to unveil the full intentions of a spambot. With a full list of possible links, we can potentially see every site that a spambot would attempt to boost. This could lead to the possibility of composing a list of spam campaign funding sources. What's peculiar, however, is observing even the links visible in Figure 5.39. We can see many domains that certainly wouldn't need to engage in illicit activity to boost its PageRank, such as 'watoday.com' and 'telegraph.co'. In Figure 5.40, we really see the format of the template. Regular sentences, but with random patches of words left in a {x | y | z} format. When the content tool is working properly is would randomly choose x, y, or z, to replace the set of words with. This gives us further insight into why the BoW feature set was so successful. The universe of words seems to be pretty small for each template, and therefore with enough posts the classifier can be exposed to nearly all of them through the training set, which would allow it to create an accurate prediction model for the rest of the posts.



Additional uncaught exception thrown while handling exception.

Original

PDOException: SQLSTATE[HY000]: General error: 2006 MySQL server has gone away: SELECT nid, cid, last_comment_timestamp, last_comment_name, last_comment_uid, comment_count FROM {node_comment_statistics} WHERE nid IN (:comments_enabled_0); Array ([:comments_enabled_0] => 1662) in comment_node_load() (line 1282 of /var/www/html/gjams.com/modules/comment/comment.module).

Additional

PDOException: SQLSTATE[HY000]: General error: 2006 MySQL server has gone away: INSERT INTO {watchdog} (uid, type, message, variables, severity, link, location, referer, hostname, timestamp) VALUES (:db_insert_placeholder_0, :db_insert_placeholder_1, :db_insert_placeholder_2, :db_insert_placeholder_3, :db_insert_placeholder_4, :db_insert_placeholder_5, :db_insert_placeholder_6, :db_insert_placeholder_7, :db_insert_placeholder_8, :db_insert_placeholder_9); Array ([:db_insert_placeholder_0] => 1 [:db_insert_placeholder_1] => php [:db_insert_placeholder_2] => %type: !message in %function (line %line of %file). [:db_insert_placeholder_3] => a:6: {s:5:"%type";s:12:"PDOException";s:8:"!message";s:271:"SQLSTATE[HY000]: General error: 2006 MySQL server has gone away: SELECT nid, cid, last_comment_timestamp, last_comment_name, last_comment_uid, comment_count FROM {node_comment_statistics} WHERE nid IN (:comments_enabled_0); Array ([:comments_enabled_0] => 1662) ";s:9:"%function";s:19:"comment_node_load()";s:5:"%file";s:54:"/var/www/html/gjams.com/modules/comment/comment.mod [:db_insert_placeholder_4] => 3 [:db_insert_placeholder_5] => [:db_insert_placeholder_6] => http://www.gjams.com/node/1662 [:db_insert_placeholder_7] => [:db_insert_placeholder_8] => 129.65.149.230 [:db_insert_placeholder_9] => 1463910571) in dblog_watchdog() (line 160 of /var/www/html/gjams.com/modules/dblog/dblog.module).

Uncaught exception thrown in session handler.

PDOException: SQLSTATE[HY000]: General error: 2006 MySQL server has gone away: SELECT 1 AS expression FROM {sessions} sessions WHERE ((sid = :db_condition_placeholder_0) AND (ssid = :db_condition_placeholder_1)); Array ([:db_condition_placeholder_0] => J9W6MYWq0tG0KSRSo842iyczpQINDtru6zY8i6P-hHs [:db_condition_placeholder_1] =>) in _drupal_session_write() (line 209 of /var/www/html/gjams.com/includes/session.inc).

Figure 5.41: The MySQL error received when trying to access any of the 'uncompiled' posts.

Also, when trying to view the post through the honeypot, we get the same MySQL

Table 5.14: Number of Entities in Top Meta Entity

Honeypot	Number of Entities
ggjx	5
gjams	20
npcagent	18

error for each post. The MySQL error is shown in Figure 5.41. We speculate that the MySQL error comes from the post being too long.

5.5.2 Top Meta Entity

As shown in Figure 5.35, the top few posting meta entities out post the rest by a significant margin. We decide to characterize the top meta entity in particular, since it represents such a large botnet and a substantial amount of the interaction as a whole with our honeypots. In Table 5.14, we observe its distribution within the three honeypots. Table 5.14 also emphasizes a point that hasn't been mentioned yet: Many of the entities within a honeypot are actually the same entity, they just weren't linked by data within that honeypot. In the cases where there is more than one entity from the same honeypot in a single meta entity, the IP additions from other honeypots are necessary to link the two intra-honeypot entities, which is why they didn't combine into one entity when the original entities were formed. Almost certainly, our meta entity list is incomplete because of the same effect; there likely exists distinct meta entities that are actually the same meta entity, but there isn't adequate access data to link them together. This supports the idea that a central intelligence with many deployed data collection points would be advantageous and effective; it allows for associations that wouldn't be possible in a decentralized setting.

This meta entity is the top posting meta entity in our set with a total of 698 posts, averaging 17 posts a day. Its total associated IP addresses is a surprisingly low 47. The meta entity with the most associated IP addresses has 263, while only posting 79 posts. This shows the diversity of our entities and the greater botnet ecosystem.

CHAPTER6

CONCLUSION

In this thesis, we set up an experiment to gather a sample set of data from the wild. We create three 'honeypot' web sites and selected a 42 day period to extract our data from. We did this in hopes of sampling a single 'spam campaign' from the spambots that interacted with our honeypots. Through the course of those 42 days, we log the network activity and the content posts on our three honeypots. From the experimental data, we form a corpus that consists of a user, content, and access table for each honeypot. The subsequent analysis is performed entirely on the corpora.

The intermediate step between the corpora and analysis was the formation of entities. We form entities as a way to model the botnets that were interacting with our honeypots, and from there apply Natural Language Processing (NLP) and Machine Learning (ML) to distinguish the most distinct set of attributes that each botnet possesses. The method we used to form the entities was intuitive: combine any IP addresses that use the same login name into one entity, and then recursively add in any other login names and their IP addresses into the entity, until every user has been accounted for and every entity's IP address collection is disjoint from every other entity. The result was a set of entities that we assume to be botnets, for each honeypot.

We then form a number of feature sets and train two different styles of classifiers with each one to determine the most unique aspects of each botnet.

Beyond contributing the corpora, the results of this experiment show that a semantic-classification model is sufficient in categorizing forum spam to their respective botnet. If nothing else, it serves as a proof of concept and a starting point for a scaled and distributed classification effort for botnets. We also discovered that some attributes were not sufficient in distinguishing one botnet from another. For example, in a domain-agnostic setting, a document's taxonomy is not a reliable trait to train a classifier on, despite being reliable within a single domain. Also, the syntactical structure of a post, while generally efficient for classifying spam vs. ham, is poor at classifying one botnet from another. We attribute the Bag of Words (BoW) attribute's success to the nature of the tools that spambots use to generate content with, and because it provides a large amount of data within its feature set. Each endpoint within a botnet is equipped with a template, provided by the botnet that is used to generate content. The template allows for different words to be used, however the word universe within a template is small. It follows that a sufficient sample size of a botnet's posts let a classifier to be exposed to its entire word set, which allows it to build an accurate model for each botnet. We also show that the majority of the network traffic that our honeypots receive comes from the same entities that post on it.

CHAPTER7

FUTURE WORK

One of the most massive botnets, ZeroAccess, is estimated to have infected 2.2 million machines [12]. When comparing to our top meta entity evaluated in 5.5.2, it's clear that this experiment only got exposed to a tiny subset of the botnet ecosystem in the wild. This thesis executed an experiment and analysis on the plausibility of characterizing forum spam into its underlying botnet. We find that the semantics of the forum spam is the most distinguishing attribute for differentiating between botnets. However, there is still a large amount of possible analysis to be done with our corpora, and further verification of if our semantic model is effective outside of our corpora.

7.1 Classification Modularization

One direction for a continuing experiment would be to modularize the semantic classifier so that it could be used on a live deployment of a web server. This would make the spam identification process that all modern web frameworks come with more sophisticated; once content is classified as spam, it could be passed to the classifier to see if it's coming from a known botnet. This would also require the classifier to be

adaptive; it would need to be retrained periodically in order to stay consistent with the changing characteristics of the spam ecosystem. Additionally, if the classifier lived independently from the server, it could mature into a central intelligence for botnets. If this were adopted by a wide range of servers and a central classifier was used, this could lead to an accurate model of the most active botnets on the web at any given time.

7.2 Multilingual Classification

Our classification process only considers posts that were made in English, however our corpora has posts in many other languages as well as multilingual posts. By extending the posts considered to include other popular languages, the usability of our classifier would be greatly extended.

7.3 Target Link Analysis

Within any entity, a number of target links may be promoted. The results from 5.3.3 suggest that for any one entity's spam posts, the links being posted are somewhat similar. Further investigation into these links could be made to find any commonalities between the links. This could lead to a source and likely suspect of the organization that is employing or running a given spam campaign, which could lead to the infiltration or prosecution of the botnet operators or their clients.

7.4 Contamination Identification

One thing that all of the endpoints of a botnet have in common is that they were all infected by the same or similar malware. One way to identify the infection strategy of

a botnet would be to cross-examine known infected endpoints. In depth diagnostics from a set of computers that are a part of the same botnet could show commonalities that reveal the way the botnet infects its victims.

7.5 Naive Bayes Classifier

As discussed in 5.3.4, the Naive Bayes classifier is a generative classifier. More research could be put in to training a more efficient Naive Bayes classifier, and then using its utility to build a more sophisticated model of the botnets. This would provide additional insight into their internal mechanisms.

BIBLIOGRAPHY

[1] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos, "An evaluation of naive bayesian anti-spam filtering", *ArXiv preprint cs/0006013*, 2000.

[2] A. A. Benczur, K. Csalogany, T. Sarlos, and M. Uher, "Spamrank–fully automatic link spam detection work in progress", in *Proceedings of the first international workshop on adversarial information retrieval on the web*, 2005.

[3] A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra, "A maximum entropy approach to natural language processing", *Computational linguistics*, vol. 22, no. 1, pp. 39–71, 1996.

[4] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python.* " O'Reilly Media, Inc.", 2009.

[5] D. Buytaert, *Drupal content management system.*

[6] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning", in *Proceedings of the 25th international conference on Machine learning*, ACM, 2008, pp. 160–167.

[7] H. Daume III, *Mega model optimization package*, 2007.

[8] P. Domingos, "A few useful things to know about machine learning", *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[9] H. Faris, K. Jaradat, M. Al-Zewairi, O. Adwan, *et al.*, "Improving knowledge based spam detection methods: The effect of malicious related features in imbalance data distribution", *International Journal of Communications, Network and System Sciences*, vol. 8, no. 5, p. 118, 2015.

[10] Google, "Google adwords keyword tool", *See https://adwords.google.com*, 2016.

[11] T. S. Guzella and W. M. Caminhas, "A review of machine learning approaches to spam filtering", *Expert Systems with Applications*, vol. 36, no. 7, pp. 10 206–10 222, 2009.

[12] J. Higgins, "Zeroaccess botnet surges", *Dark Reading*, 2012.

[13] P. Kolari, A. Java, T. Finin, T. Oates, and A. Joshi, "Detecting spam blogs: A machine learning approach", in *Proceedings of the National Conference on Artificial Intelligence*, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, vol. 21, 2006, p. 1351.

[14] O. LLC, "Alchemyapi", *Available online http://www. alchemyapi. com/(accessed December)*, 2009.

[15] X. Mertens, *All cve details at your fingertips*.

[16] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes-which naive bayes?", in *CEAS*, 2006, pp. 27–28.

[17] J. Nazario, "Phoneyc: A virtual client honeypot.", *LEET*, vol. 9, pp. 911–919, 2009.

[18] E. Peter and T. Schiller, "A practical guide to honeypots", *Washington Univerity*, 2011.

[19] M. Porter, "The porter stemming algorithm, 2005", *See http://www.tartarus.org/˜martin/PorterStemmer*,

[20] N. Provos, "Honeyd-a virtual honeypot daemon", in *10th DFN-CERT Workshop, Hamburg, Germany*, vol. 2, 2003, p. 4.

[21] F. Sebastiani, "Machine learning in automated text categorization", *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[22] Y. Shin, M. Gupta, and S. A. Myers, "The nuts and bolts of a forum spam automator.", in *LEET*, 2011.

[23] B. Stone-Gross, T. Holz, G. Stringhini, and G. Vigna, "The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns.", *LEET*, vol. 11, pp. 4–4, 2011.

[24] Symantec, "Internet security threat report", vol. 21, 2016.

[25] A. Thomason, "Blog spam: A review.", in *CEAS*, 2007.

[26] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network", in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, Association for Computational Linguistics, 2003, pp. 173–180.