A NATURAL LANGUAGE PROCESSING ANALYSIS

PETER MASON
SPRING 2017

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

John O’Hara
Associate Research Engineer, Penn State ARL
Thesis Supervisor

John Hannan
Associate Department Head
Honors Adviser

* Signatures are on file in the Schreyer Honors College.
ABSTRACT

In an era where “Big Data” pervades nearly every industry with the hope of gleaning new insights or ironing out inefficiencies, one often-overlooked place is the self-proclaimed “front page of the Internet.” The news aggregator and discussion forum Reddit (www.reddit.com) boasts several hundred million monthly users, and the documented interactions between so many strangers is a potential goldmine waiting to be tapped.

In this thesis we use natural language processing, a burgeoning field in computer science, to explore and analyze the comment corpus of all Reddit users between July 2013 and May 2015. We first look into Reddit-specific phenomena, particularly the frequency and distribution of textual memes and reposted comments, and find that some memes maintain their influence and popularity throughout the two-year span while variations develop and persist within smaller online communities. The rate of reposts—unoriginal content—increases over that period as well, both in absolute numbers and when adjusting for the growing number of users.

We next study privacy and sharing habits of Reddit users, looking at how often names, addresses, and phone numbers are given out. Distinguishing between all names, addresses, or phone numbers and personal information proves a more intractable task, though some heuristics help us approximate an upper bound for all three categories that are shared throughout the two years.

Finally, we explore some patterns relevant to other forms of social media. We analyze the structure of high-scoring comments and overtly political comments, and using some machine learning we construct predictive classifiers for both areas. We ultimately find that while
sentence structure and named entity categories certainly have some impact on whether a comment scores highly or is political, we are unable to reason out why that is the case.
# TABLE OF CONTENTS

List of Figures .............................................................................................................. iv

List of Tables ................................................................................................................ v

Acknowledgements ........................................................................................................ vi

1 Introduction and Motivations ...................................................................................... 1
   Reddit Phenomena ...................................................................................................... 1
   Privacy and Sharing ................................................................................................... 2
   Social Media Patterns ............................................................................................... 3

2 Natural Language Processing Overview ........................................................................ 5
   A Brief History of NLP ............................................................................................... 5
   NLP Terminology and Capabilities ............................................................................. 6

3 Dataset and Technology Stack .................................................................................... 10
   Comment Format ....................................................................................................... 10
   Technology Stack Overview ..................................................................................... 12
   Natural Language Toolkit (NLTK) ............................................................................. 13
   Apache Hadoop .......................................................................................................... 14
   Amazon Web Services (AWS) and Elastic MapReduce .............................................. 16

4 Approach and Methodology ....................................................................................... 17
   Reddit Phenomena Outline ...................................................................................... 18
   Privacy and Sharing Outline ..................................................................................... 20
   Social Media Patterns Outline .................................................................................. 21

5 Results and Analysis .................................................................................................. 24
   Summary Statistics ..................................................................................................... 24
   Reddit Phenomena Analysis ...................................................................................... 27
   Privacy and Sharing Analysis ..................................................................................... 31
   Social Media Patterns Analysis .................................................................................. 34

6 Opportunities for Further Study ................................................................................ 41

Appendix A ..................................................................................................................... 43

Bibliography .................................................................................................................. 57
List of Figures

Figure 1: Reddit Flair Example.................................................................12
Figure 2: Hadoop MapReduce Visualization...........................................14
Figure 3: Total Points across Comments, by Month..................................24
Figure 4: Number of Comments, by Month.............................................24
Figure 5: Average Points per Comment, by Month....................................25
Figure 6: Number of Users, by Comments per Month...............................26
Figure 7: Navy Seal Meme Frequency, 7/2013 - 5/2015..............................27
Figure 8: Navy Seal Copypasta Variants.................................................28
Figure 9: Total Number of Reposted Comments......................................29
Figure 10: Normalized Reddit Reposts....................................................30
Figure 11: Shared Personal Information on Reddit....................................34
LIST OF TABLES

Table 1: Reddit Comment Attributes .................................................................................. 11
Table 2: Most Informative Non-Comment Features ........................................................... 36
Table 3: Most Informative Structural Features .................................................................. 37
Table 4: Most Informative Political Structural Features .................................................. 39
Acknowledgements

I would like to thank John O’Hara, who has been with me since my first internship at Penn State’s Applied Research Lab, and who has instilled lifelong programming practices and habits in me over the past four years.

I would also like to thank John Hannan, my honors adviser and professor for several courses. Dr. Hannan has always held me to a high standard throughout my college education and impressed me with his understanding of various computer science topics.
1 Introduction and Motivations

When the amount of text, image, and video data being produced globally continues to grow exponentially, it begs the question of why one should examine any specific subset of that data. The pseudo-anonymous social media site Reddit (www.reddit.com) has a base of 234 million active monthly users, and is the fourth most visited website in the U.S. and sixteenth most visited in the world according to the web analytics company Alexa (Gallo). While Reddit can be viewed as “just” a news aggregator or discussion forum, we can analyze many patterns that can be extrapolated elsewhere. We now describe the three main categories on which we will concentrate our efforts. To further outline our work, we then discuss relevant natural language processing techniques, the Reddit comment corpus, and the list of technologies used before diving into our methodology and analysis of the three categories below.

Reddit Phenomena

Some of the lowest-hanging fruit for natural language processing on Reddit may also be some of the most bizarre. Reddit is home to many corny jokes and puns, a constant influx of memes, countless reposts. Still, there are patterns to be found that could be crucial for corporate marketing or PR departments. Specifically, we look for the frequency and variance of certain memes. The psychological questions for what makes memes popular is beyond the scope of this paper, but we will examine the relevant correlations. That is, we study whether variations of memes localized to certain communities or subreddits, and how popular these variations are, without knowing why they exist in the first place. Furthermore, we look at pre-identified memes and examine their use over time, and the rate at which memes go in and out of fashion. For one
particular meme known as the “Navy Seal copypasta,” we find that its usage remains close to constant throughout the two-year span, with roughly two hundred occurrences per month, while dozens of less common variations spring up during that time. This has several real-world impacts, especially for the PR departments of companies desperate to attract the attention of an always-online generation. Making old references or jokes that died out mere months ago could easily turn a company into a temporary laughingstock, while other memes prove to be more reliable staples.

Additionally, we look for patterns for reposts—the repeated posting of comments with unoriginal content—in both popular and lesser-known subreddits. We attempt to determine which reposts will hit the front page and have their fifteen minutes of Internet fame. Being able to time the posting of content—even unoriginal content—for maximum impact could be of interest to practically everyone. We find that reposting has significantly increased during our time period, from 80,215 reposts in July 2013 to 206,748 reposts in May 2015, representing a 157.7% increase. When we adjust for the increase in Reddit’s user base over that time, we still find a 65% increase, which points to a relative lack of new ideas and content as Reddit continues to grow. Either we were unable to identify any meaningful trends for predicting either the occurrence reposted comments or how well the reposts scored.

**Privacy and Sharing**

The Nobel Prize-winning economist Paul Krugman said in 1998 that "the growth of the Internet will slow drastically, as [it] become apparent: most people have nothing to say to each other!" While hindsight is 20/20, the extent to which people will share the most intimate details
of their lives is still an open question. We use natural language processing to search for common identifying characteristics—specifically names, addresses, and phone numbers—to analyze how frequently complete strangers open themselves up to the online public. It may also be interesting to look at the sharing differences between smaller, close-knit communities and the much larger default subreddits. These sharing habits could also reflect upon what Reddit users consider pseudo-anonymous. They may be far more willing to share their name than their address, but resourceful adversaries could mine comment histories to single out identities from very little explicitly personal information. Knowing these habits could serve as the basis for education about online interactions, applicable to all ages.

With that in mind, natural language processing techniques are still fairly limited for identifying what information is actually personal. Using a combination of regular expressions and named entity recognition, we establish an upper bound of 21,000 names, 300 addresses, and 453,000 phone numbers shared.

**Social Media Patterns**

Our final but broadest motivation is to find patterns, associations, and correlations within the Reddit comment corpus that can extend to other social media. We first look at the natural language patterns intrinsic to high-scoring comments. There may not be a magic formula for getting thousands of anonymous people to like what you say, but NLP analysis can result in some starting guidelines. On the other side of the coin, we can determine whether there are any warning signs or giveaways for low-scoring comments. We also consider some outside factors, including the posting subreddit, the users themselves, and other information. We find that the
metadata comprising these outside factors is some of the most powerful information we have, with comments in certain subreddits (DotA2, nfl, hockey, nba, and others) being disproportionately low-scoring. We also find linguistic patterns indicative of high- and low-scoring comments, though they are somewhat less powerful.

We analyze relevant patterns for political comments as well. In an extremely politically polarized climate, focusing on patterns similar to many, if not all, political discussions could yield interesting results. We glean some insights into how political and apolitical comments are grammatically structured, and list the relevant structures and patterns in Chapter 5.
2 Natural Language Processing Overview

Natural language processing (hence referred to as NLP) is a rapidly growing field in computer science and a hammer with which many, many things become nails. NLP seeks to structure and process human speech ("natural" language) that has already been converted to text. Note that NLP is separate from voice recognition, which aims to convert sounds and spoken words into text. The two fields are closely related, especially with the rise of personal assistant software such as Siri and Cortana that utilize both technologies, but this paper is concerned only with the former. We will now provide a brief history of NLP to paint a clearer picture of its direction and breadth and then explicitly talk about its capabilities.

A Brief History of NLP

Natural language processing began in the 1950s, when Alan Turing published the paper "Computing Machinery and Intelligence." Turing's paper described the now-popular Turing Test, a black-box experiment testing if a human could tell the difference between a computer and a second human based solely on their textual interactions (Moor). In one of the first actual natural language processing tasks, the 1954 Georgetown-IBM experiment sought to automatically translate dozens of Russian sentences into English (Hutchins). The experiment was successful, to the extent that the authors claimed that the entirety of natural language processing would be a solved problem within five to ten years. However, English and many other spoken languages have proven themselves ambiguous and resistant to more popular and grounded linguistic theories such as context-free and transformational grammars. The advent of machine learning in the 1980s sparked a follow-up NLP revolution, as more statistical
approaches involving part-of-speech tagging and hidden Markov models found greater success (Berger). Current natural language processing continues to employ probabilistic techniques, machine learning, and deep learning to accomplish a wide variety of tasks.

**NLP Terminology and Capabilities**

Natural language processing is a broad field, ranging from pure interpretation of natural language into computer-readable formats to text prediction and analysis. We now define a list of NLP vocabulary that will be used throughout our analysis:

- **Corpus** - A structured and usually large set of text. In this paper, our singular corpus comprises all Reddit comments from July 2013 to May 2015.
- **Token** - A sequence of characters that are grouped together for processing. These sequences are most commonly words, but can also be punctuation or whitespace.
- **POS-tagging** - Part of speech tagging involves associating every word in a sentence or text with its part of speech. Common POS tags include DT (determiner), NN (singular or plural noun), JJ (adjective), and VB (base form of a verb).
- **Regular expression** - A sequence of characters that define a pattern. Regular expressions (regexes) are frequently used to search for certain kinds of strings.
- **CFG** - Context-free grammars define a language by using predefined rules and terminal symbols. One such rule in NLP might be a sentence S comprises a noun phrase and a verb phrase.
- **Chunking** - The process of using POS-tagged text to build up the structure of a sentence, such as recognizing "under (IN) the (DT) bridge (NN)" as a prepositional phrase (PP).
Natural language processing is a broad field, ranging from pure interpretation of natural language into computer-readable formats to text prediction and analysis. Sentence parsing, POS tagging, and chunking all fall under syntax tasks, and we can build upon those techniques to accomplish entity recognition (finding proper nouns and entities), sentiment analysis, and other semantic goals.

In this paper, we will use several of the above techniques. We utilize use regular expressions primarily for searching tasks, like finding addresses and phone numbers. Regular expressions define a particular language, or a set of acceptable strings. For instance, phone numbers could be present in many similar yet slightly different formats. They could have parentheses, dashes, spaces, some combination of two or all three, or none of the above. With that in mind, we want to construct a language that encompasses all of these options, and regular expressions allow us to do just that.

In order to take advantage of named entity recognition, our most powerful natural language processing technique, we must first have all the necessary building blocks in place. We begin by tokenizing a sentence, but quickly run into difficulties due to the frequency of unusual characters and combinations of characters present in online text. We therefore abandon the common regular-expression-based tokenizer in favor of the more specialized Penn Treebank (PTB) tokenizer, a large open-source project and standard. We look to the Penn Treebank specifically due to its annotations centered on phrase structure (rather than dependency structure) since we can hook into chunkers more easily later on. We use an open-source corpus of online chat logs (a set of text very similar to Reddit comments) that contains both raw and pre-tagged versions, and thus are able to train the PTB tokenizer on a subset of that data and confirm that it
performs accurately. After the successful tokenization of the text, we then use a backoff POS-tagger, which attempts to link words to POS tags according to a database, then failing that backs off to a regular expression approach. NLTK provides the model for the backoff tagger for us, but we again train it on the chat corpus. We additionally write the attributes of both the tokenizer and the tagger to file to avoid retraining every time we need those functions.

Our second main NLP tool is named entity recognition (NER). Once we build up the structure of a sentence by tokenizing the sentence, tagging each token as a part of speech, and then chunking tagged tokens together, we can finally link words or phrases to some greater meaning. For example, named entity recognition allows us to associate “United States” as a geopolitical entity rather than just an adjective and a plural noun shoved together. Chunkers create a tree structure from the tagged tokens, at which point we use CFG tools to parse and analyze that structure. We take advantage of the power and flexibility of NER to identify names and more thoroughly examine high-scoring and political Reddit comments in our analysis.

We now give an example of the tokenization-tagging-chunking process that we repeat numerous times and is critical in named entity recognition. We start with the sentence “We are Penn State.” We tokenize the text, separating it into a list of words, spaces, and punctuation. We then tag each token with a part of speech, now creating a list of (word, POS) pairs. Finally, we chunk the text, grouping the tagged tokens into larger entities. In this case, we form an organization chunk for “Penn State,” and a sentence chunk for the entire text.

- Raw text: “We are Penn State.”
- Tokenized text: ['We', 'are', 'Penn', 'State', '.']
- Tagged text: [('We', 'PRP'), ('are', 'VBP'), ('Penn', 'NNP'), ('State', 'NNP'), (',', ',')]
Here, the relevant tags are PRP: personal pronoun, VBP: singular non 3rd person present tense verb, NNP: singular proper noun, and the period tag.

- Chunked text:

(S
  We/PRP
  are/VBP
  (ORGANIZATION  Penn/NNP
                  State/NNP)
  .)

- Using the same tags as before, we create a tree structure with “Penn” and “State” falling under the ORGANIZATION branch and everything falling under S (sentence).
3 Dataset and Technology Stack

Even with Moore’s Law still holding strong, the rapidly growing mountain of data—2.5 million terabytes of information generated in 2016 alone (InfoSphere)—forces us to pick and choose our analyses carefully. With that in mind, Reddit is a source for hundreds of millions of text comments from millions of users, with wildly varying opinions, quality, and length. The wide distribution of such qualities lends itself well to NLP algorithms, which often require one subset of data to train a model and another subset to test it. We then expand upon the technologies we use to leverage the NLP techniques described in the previous chapter.

The dataset itself was made public in July 2015 by Reddit user u/Stuck_In_the_Matrix, and remains available at https://archive.org/details/2015_reddit_comments_corpus. We chose to analyze a large subset of the data, specifically all comments made between July 2013 and May 2015, due to local hardware limitations.

Comment Format

The structure of the comment corpus is fairly straightforward, but requires some explanation nonetheless. From the initial corpus download, comments are divided by year and then subdivided by month. All comments made in a particular month are contained in one JSON file, which is a widely used human- and computer-readable format. Each comment is formatted as a one-line JSON object with a set of attributes that we define and explain in the table below.
### Table 1: Reddit Comment Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>subreddit</td>
<td>Name of the subreddit in which the comment was posted</td>
<td>worldnews</td>
</tr>
<tr>
<td>author</td>
<td>The user who posted the comment</td>
<td>gghy2</td>
</tr>
<tr>
<td>score</td>
<td>Score for the comment (upvotes minus downvotes)</td>
<td>1</td>
</tr>
<tr>
<td>ups</td>
<td>Number of upvotes</td>
<td>1</td>
</tr>
<tr>
<td>downs</td>
<td>Number of downvotes</td>
<td>0</td>
</tr>
<tr>
<td>name</td>
<td>ID in hexadecimal for the author</td>
<td>t1_catgbbk</td>
</tr>
<tr>
<td>id</td>
<td>Optional ID for more comments under this one</td>
<td>null</td>
</tr>
<tr>
<td>link_id</td>
<td>ID in hexadecimal for the comment itself</td>
<td>t3_1hdlt8</td>
</tr>
<tr>
<td>subreddit_id</td>
<td>ID in hexadecimal for the subreddit</td>
<td>t5_2qh13</td>
</tr>
<tr>
<td>parent_id</td>
<td>ID in hexadecimal for the parent comment</td>
<td>t3_1hdlt8</td>
</tr>
<tr>
<td>created_utc</td>
<td>Time in UTC when the comment was posted</td>
<td>1372637149</td>
</tr>
<tr>
<td>edited</td>
<td>Boolean value for whether the comment was edited</td>
<td>false</td>
</tr>
<tr>
<td>score_hidden</td>
<td>Boolean value for whether the comment score is hidden</td>
<td>false</td>
</tr>
<tr>
<td>gilded</td>
<td>Number of times the comment received Reddit Gold</td>
<td>0</td>
</tr>
<tr>
<td>distinguished</td>
<td>a moderator, or an administrator</td>
<td>null</td>
</tr>
<tr>
<td>archived</td>
<td>Boolean value for whether the comment was archived</td>
<td>true</td>
</tr>
<tr>
<td>controversy</td>
<td>A composite value given by Reddit's algorithm to determine controversy</td>
<td>0</td>
</tr>
<tr>
<td>retrieved_on</td>
<td>Time in UTC when the comment was retrieved using Reddit's API</td>
<td>1430626994</td>
</tr>
<tr>
<td>author_flair_css_class</td>
<td>The CSS class of the author's flair (if applicable)</td>
<td>null</td>
</tr>
<tr>
<td>author_flair_text</td>
<td>The text of the author's flair (if applicable)</td>
<td>null</td>
</tr>
<tr>
<td>body</td>
<td>The actual text that the user wrote in the comment</td>
<td>Test post please ignore</td>
</tr>
</tbody>
</table>

A few of the comment attributes may not be immediately apparent and merit further explanation. Archived comments are those that have existed long enough to be archived by Reddit, meaning that they can no longer be upvoted, downvoted, or replied to by anyone. The “gilded” attribute refers to Reddit Gold, a virtual prize of sorts that users can purchase for $4 USD and reward to outstanding comments or posts for a month. Finally, “flair” is a kind of label attached to users within a subreddit, and can be used to express preferences such as a user’s favorite team or player within a sports subreddit. “author_flair_text” thus refers to the content of
the flair itself (e.g. “Steelers”), while “author_flair_css_class” refers to the aesthetics and design the user has chosen to display that flair.

![Reddit Flair Example](image)

In Figure 1, u/HuntertheDragoon displays a Ravens flair, while the associated CSS class governs how the Ravens symbol actually displays on Reddit.

For the purposes of this thesis, we limit our scope to the author, subreddit, score, gilded, controversiality, and body attributes of Reddit comments.

**Technology Stack Overview**

As natural language processing and "big data" continue to grow, so do the number, quality, and documentation of relevant technologies. We use several popular open-source tools in our analysis, and now outline our decision-making process for choosing this particular technology stack, which consists of the Python Natural Language Toolkit, Apache Hadoop, and AWS Elastic MapReduce.
Natural Language Toolkit (NLTK)

Python's Natural Language Toolkit (NLTK) is a set of libraries and modules developed specifically for natural language processing. As its chief developer Steven Bird writes in *Natural Language Processing with Python*, NLTK has support for "classification, tokenization, POS-tagging, and parsing (Bird)." The toolkit is easily installable via the Python package manager pip. NLTK comes loaded with several pre-tagged corpuses, including the well-known Brown Corpus and the Chat Corpus, a collection of discussions from an online Firefox forum. Due to the inherent similarities between anonymous comments across the Internet, we use the Chat Corpus to help train our classification models during our analysis. NLTK is in continuous development and currently is one of the most documented NLP platforms, and supports both Python 3.7 and 3.4 onward. Some competitors offer features and tools absent in NLTK, such as Stanford CoreNLP's extensibility and Apache OpenNLP's machine learning capabilities, but Python's quick development cycle and the toolkit's ease of use and built-in data sets tilted our decision in NLTK's favor. Additionally, we take advantage of Python's json module for easy interaction with the corpus text files. With three lines of code, we can read from a file and convert the JSON object into a dictionary, one of Python's default data types, as seen below.

```python
commentLine = open(fileName).readLine()
commentDict = json.loads(line)
commentText = commentDict['body']
```

The “commentDict” object is the dictionary containing all of a comment’s attributes as described in the previous section, while “commentText” is a string containing the actual written (typed) comment contents.
We choose to develop primarily in Python 3.6 because of its default Unicode string encoding and tuple unpacking feature. Virtually all other aspects of Python 3.6 and its main alternative, Python 2.7, are identical for the purposes of this thesis.

**Apache Hadoop**

Even with efficient analyses of raw text, the sheer size of the Reddit comment corpus forces us to look past the paradigm of working solely on one machine. For the twenty-three month span of comments we analyze between July 2013 and May 2015, the total corpus size is roughly 700 gigabytes. Apache Hadoop, an open-source big data technology, provides a framework for easily performing distributed and parallel computation. Hadoop, which natively supports Python, is composed of a redundant distributed file system, known as HDFS, and a system that processes the stored data in a parallel manner before compiling and reducing everything to a singular result. The latter half of Hadoop is known as MapReduce, named for its

![The MapReduce Pipeline](image)

*A mapper receives (Key, Value) & outputs (Key, Value)*
*A reducer receives (Key, Iterable[Value]) and outputs (Key, Value)*

*Figure 2: Hadoop MapReduce Visualization*
"mapping" and "reducing" subroutines. Thanks to Hadoop's ecosystem of surrounding projects, it presents itself as a natural choice for NLP processing that can be written as batch tasks.

As an example, we provide sample code for our first and simplest task: calculating the total number of comments.

**Mapper.py**

```python
commentCount = 0
for line in file:
    commentCount+=1
print(commentCount)
```

**Reducer.py**

```python
totalCommentCount = 0
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # convert commentCount to an int
    try:
        commentCount = int(line)
    except ValueError: # count was not a number, so ignore
        continue
    totalCommentCount += commentCount
print("Total comment count is: ", totalCommentCount)
```

We have hidden the file path and list of imported modules in this example, but otherwise the files are complete. All mappers running in parallel simply print their results to the console, which then is piped into the input of the reducers.
Amazon Web Services (AWS) and Elastic MapReduce

One of the easiest and cheapest ways to gain access to a computing cluster is to take advantage of cloud. Amazon Web Services (AWS) is currently the largest cloud computing platform with a forty percent market share for all public IaaS (Infrastructure as a Service) and PaaS (Platform as a Service) (Gallo). AWS offers credit for educational use and furthermore has some of the cheapest computing costs per hour, making it an appealing option for our purposes.

Elastic MapReduce (EMR) is a service within AWS that uses Hadoop to process large quantities of data and quickly scale up as needed. EMR therefore allows us to construct a fairly robust development pipeline for work where parallelization is necessary. We can create a local pseudo-cluster (an HDFS on one machine that acts like it exists on multiple machines), write a Hadoop job to test on a small amount of data, then run the job on EMR across multiple Amazon EC2 instances to ultimately complete a task in hours instead of days.
4 Approach and Methodology

Before beginning our analysis of the Reddit comment corpus, we first outline more precisely what we are analyzing, define the approach we take towards producing an answer or solution, and explain why we use that particular approach. We will repeat this process for each major section of our analysis, beginning with summary statistics and moving on to Reddit phenomena, privacy and sharing tendencies, and other social media patterns.

Summary Statistics Outline

In order to make reasonable assumptions and accurate conclusions about our time-related analyses, namely the frequency of memes and reposts and the occurrences of shared personal information, we will need to look briefly at the distribution and frequency of all Reddit comments first. With that motivation in mind, we find the total number of comments made per month, the average number of points per comment by month, and the number of comments per unique (active) user. We accomplish the first two tasks by simply incrementing a counter per comment (and keeping a running sum of the total score) in our mapping function of Hadoop, then add all the results together in our reduce function. We then divide the total score per month by the number of comments per month to get the average score per comment. To find the number of active users, we have each node in the Hadoop cluster again iterate over the comments, this time periodically writing a dictionary of all (unique user, number of comments by user) pairs to a local file. Upon completion, we combine (reduce) these lists to obtain a master list of the number of comments made by each user.
The near-monotonically increasing number of comments per month that we find means that we will have to normalize our results for the frequency of memes and reposts, which we choose to do by dividing all results by the number of comments made in our first month (July 2013). We then find that the average (mean) comment score remains close to constant throughout the two years, at roughly 5.75 points. From here, we can make the reasonable assumption that there is no correlation between the time of the comment’s posting and its score, which is helpful when identifying what factors contribute towards high-scoring comments in our social media patterns section. Finally, we find a wide distribution of the activeness of users, with the vast majority making under ten comments per month but some making over ten thousand. This is more useful information, as we take advantage of unusual commenting habits to extract identifying information for high-scoring and political comments.

**Reddit Phenomena Outline**

With the summary statistics conclusions in mind, we proceed to the two main goals of studying Reddit-specific phenomena: the distribution and variety of memes and the frequency of reposted comments.

We quickly run into one immediate roadblock when studying memes via natural language processing; the majority of memes are image-based, often with a re-used image that defines the meme “category” and overlaid lines of text at the top and/or bottom of the image. Because we cannot easily process or analyze these memes without delving into other computer science topics like image recognition or computer vision, we look at the remaining text-based options instead. Due to time constraints, we limit our analysis to one meme and all of its spin-offs. We predict
more variation in longer memes, and therefore choose the lengthiest yet still common text-based meme we can find. A brief Internet search leads to us to a meme called the “Navy Seal copypasta” or alternatively the “gorilla warfare copypasta.” This meme takes the form of a 300-word rant, littered with misspellings, in which the user pretends to be a Navy Seal and mockingly threatens the target. The rant is copied and pasted into seemingly arbitrary online situations, thus the term “copypasta.”

We now recall our first original goal of studying the popularity of memes and if or how localized variations develop over time. With our meme of choice in hand, we first look at the number of occurrences of the original Navy Seal copypasta between May 2013 and July 2015. We accomplish this via simple string matching, using regular expressions to account for very small aberrations like letter case, whitespace, and contractions. We then proceed to look at the copypasta variations with a combination of more complex regular expressions and chunking, which allows us to match the general grammatical structure of the meme without needing to match specific words.

Moving on to the frequency of reposts, we start with our definition of repost: an unoriginal comment, or more specifically a comment whose entire body is already present earlier in the corpus. We acknowledge that not all identical posts are intentional reposts, as many greetings or niceties are limited in variety. We account for these unintentional copycats by limiting our scope to comments that are twenty characters or more in length. We furthermore ignore all whitespace before and after the comment to ignore minor formatting differences.

To actually count the number of reposts, we create a priority queue with all (comment, repost count) pairs to count the number of times each comment is posted. If we come across a comment that is not present in the priority queue, we add a new entry with its count set to one. It
is impractical to store all comments in memory, so once the priority queue reaches a large enough size, we treat it like a cache and begin writing the dictionary to a local file. We then keep the most-reposted comments in the queue (with the repost count as the priority), and for each new comment we first check the priority queue and failing that the file, then update the file and queue as needed. We conclude our repost search by adding all the repost counts together across all months in our reduce function.

Privacy and Sharing Outline

Here we outline how and why we attempt to find all occurrences of personal names, addresses, and phone numbers in Reddit comments. Our overarching goal is to identify any kind of personal information sharing, but many such instances do not have any consistent form or linguistic structure, and thus are difficult to pinpoint. A user could share his or her pets, hobbies, or profession, but might do so in a way that is vastly different from someone else. For example, the sentences “I program for a living” and “I am a software developer” say almost exactly the same thing, but have considerably different syntax, and furthermore may be difficult to programmatically categorize “software developer” as a profession. On the other hand, some other kinds of personal information generally have fewer such possibilities, such as “My name is…” or the format of an address or phone number. We therefore choose the latter three pieces of information to analyze for their relative ease of discovery.

We employ similar approaches to identify all three types of information. Steven Bird writes in *Natural Language Processing with Python* that classifying names and addresses in text is a solved problem with named entity recognition (Bird), which is the main tool we will use in
our analysis. The one significant caveat is that while NER is useful at identifying general names, locations, and other entities, it is not especially good at linking them to a user or the first person perspective. Surprisingly, we were unable to find any public research related to that problem, and instead rely on more fundamental NLP techniques. We still use named entity recognition to pull out names and addresses (via the tokenization-tagging-chunking process described in Chapter 2), and add regular expressions to match for several key phrases, including “My name is” and “I live at.” We track an instance of a name being shared when we find the matched pattern (e.g. “My name is”) followed by a PERSON entity (e.g. “Alex”). We similarly track addresses with the appropriate pattern and CD (number) and LOCATION/GPE (geo-political entity) chunks. The structure of phone numbers is even more rigid, so regular expressions are the perfect tool in this situation. We match for patterns of ten to eleven digits, with optional parentheses, spaces, dashes, and country code. We list the regex phrases for names and addresses, as well as the single phone regular expression, in the relevant part of Appendix A.

**Social Media Patterns Outline**

Our third and final goal is to identify high-scoring and political comments, as well as any associated patterns. To that end, we naturally want to be able to predict whether a comment is high-scoring or political, and therefore need to construct a predictive model using the Reddit comment corpus data. There are two such machine learning models with substantial support in Python, the Naïve Bayes classifier and the Maximum Entropy classifier (Berger). Both models require training sets of labelled data, or in other words a list of comments tagged as either high-scoring or low-scoring (political or apolitical for the latter predictive model). We additionally
must extract features from comments (like comment length or number of nouns), which are used by the classifiers to find correlations between feature values and labels. The Maximum Entropy model is a generalization of the Naïve Bayes model, where features do not have to be tied to one label and we do not work under the assumption that features are independent of each other. However, given the nature of our binary labels (high-scoring vs. low-scoring and political vs. apolitical) and our choice of features, the Naïve Bayes classifier is a better fit for our purposes.

We are then left with the task of tagging comments with labels to eventually train the classifier. For the question of identifying high-scoring comments, this is easy enough. We define high-scoring comments as those with scores above one thousand, representing less than one percent of all comments. Note that we have numerous features, which are described and explained in more detail in Chapter 5. We gather ten thousand each of labelled high-scoring and low-scoring comments—enough to give exposure to all permutations of features and labels—and train the classifier with that data. From here, we use NLTK to analyze the model’s predictive power and find the most powerful (informative) features.

We repeat this process for our second task of identifying political comments, with the one added difficulty of somehow labelling ten thousand political and apolitical comments. As a solution to this roadblock, we choose to define all comments in the politics subreddit as political. Similarly, we gathered comments from the science, askscience, and AskHistorians subreddits, which are all heavily moderated and remove comments involving contemporary politics, to create our apolitical set. We admit that this solution is not perfect, as not all comments in the politics subreddit are necessarily political, nor is the content of three specific subreddits representative of all apolitical comments on Reddit. After collecting ten thousand comments for
each label, we extracted similar features as with the high-scoring comments and tested their effects on classification.
5 Results and Analysis

With our outlined goals of investigating Reddit-specific phenomena, privacy and sharing tendencies, and other social media patterns, we now document what techniques we use and the following results. All figures are plotted with Python’s matplotlib library unless otherwise specified.

Summary Statistics

We begin with an analysis of the corpus itself. Some patterns found via NLP techniques may be explained by simple changes to Reddit’s active user base, so it is in our best interest to account for these changes and any resulting confounding variables. We first find the total number of comments made by month and the total number of points from summing the scores of all comments, as seen in Figures 3 and 4 respectively.
Both bar charts show a trend of generally increased activity, which could mean Reddit is gaining more users, existing users are commenting and upvoting more frequently, or some combination of the two. We know from Reddit’s own traffic statistics that the monthly active user count increased from 85.9 million in May 2013 to 163.97 million in July 2015, which strongly confirms the first hypothesis. To confirm or deny the possibility of users upvoting more frequently, we can look at average comment scores by simply dividing the total number of points per month by the number of comments per month.

![Average Comment Scores](image)

**Figure 5: Average Points per Comment, by Month**

The distribution of comment scores across the twenty-three month period is close to uniform, which implies that the correlation between number of comments and total points is very high, and furthermore that the driving factor behind the comment and point growth is the increase in active users.
To examine just how active those users are, we can look at the number of comments individual users made each month. Figure 6 shows how many users made a given number of comments.

![User Activeness Distribution](image)

Figure 6: Number of Users, by Comments per Month

A majority of users made under ten comments per month, while a small but still sizeable portion of Reddit users made thousands or even tens of thousands per month. A closer analysis tells us that the median of this distribution is 4.0, while the mean is 21.3 comments per month. This confirms that the distribution is heavily skewed to the right, with the Reddit “whales” pulling the mean well above the median.
Reddit Phenomena Analysis

We begin our natural language processing analysis with two Reddit-specific patterns: the distribution of a handful of memes throughout our twenty-three month span and the frequency of reposted comments across Reddit.

Although we are limited to natural language processing and the majority of Internet memes are presented as images, we are still to identify the many text-based counterparts. Here we analyze the most common copypasta from 2013-2015, the “Navy Seal copypasta.” This copypasta occurred 5,114 times over the two years, which we obtained by looking for near-exact matches throughout the comment corpus. It is also interesting to note the distribution of the meme across time and how its popularity is somewhat stable, even when adjusted for Reddit’s increasing user base.

![Figure 7: Navy Seal Meme Frequency, 7/2013 - 5/2015](image)
We found an additional 7,698 variants of this particular copypasta that all mirror its grammatical structure. These variants are often have themes based on their parent subreddit. The vanilla Navy Seal copypasta search was performed using a straightforward regular-expression approach, while the variants used a combination of regular expressions and fuzzy string matching to find approximate matches for certain sections of the skeleton/template of the original. As seen in Figure 8, a large minority of the variants occurred less than twenty times each.

![Reddit Navy Seal Copypasta Variants](image)

*Figure 8: Navy Seal Copypasta Variants*

We found that for each of the common variants, the copypastas matched exactly. This would imply either that there is either some database or resource for users to quickly look up specific copypastas, or that small numbers of users are responsible for posting a specific variant
many times. In fact, inspection of the variants found that a Reddit bot (an automated script) commented the vast majority of the social justice copypasta version.

Within Reddit, reposts are often cited as a growing problem and the cause of the death of originality. With that in mind, we now examine the frequency of reposts from July 2013 to May 2015, specifically looking for the absolute increase or decrease in reposts and that change relative to the increase in the total number of comments.

In order to better parallelize this task, we only count a comment as a repost if it occurs within the same month as the original. Since many one- and two-word comments are identical simply by nature of having fewer total permutations, we also created a lower bound of twenty characters for comments to count as a reposts.

![Figure 9: Total Number of Reposted Comments](image-url)
The trend for the absolute number of reposts is absolutely an increasing one; July 2013 recorded 80,215 reposts compared to May 2015’s 206,748. We then normalized the results, framing all numbers in terms of the number of comments in July 2013.

With the exception of some major outliers in late 2013 and early 2014, we still have a generally increasing trend, albeit a much shallower one. We have no explanation for what caused the temporary spike in reposts starting in October 2013, and leave that speculation for future research. We conclude that complaints about reposts are well-founded, though they may be unfortunate but inevitable growing pains for a pseudo-anonymous social media site like Reddit.
Privacy and Sharing Analysis

Our second objective explored the extent to which users are willing to share information with others—albeit anonymously. We chose to focus on users sharing names, addresses, and phone numbers. These three pieces of information generally take a more rigid format, as in [first name] [last name] for an identifying name, [number] [street] [city] for an address, and some combination of parentheses, dashes, spaces, and ten to eleven digits for phone numbers. These formats correspond nicely to certain tags used for named entity chunking, namely PERSON, CD (cardinal number), FACILITY (includes streets), and GPE (geopolitical entity, including cities). All chunking and POS tags are defined in accordance with the Penn Treebank Project, a large annotated text corpus—that serves as a standard in the natural language processing community.

We tried two different approaches to identify instances of users sharing their own names. We first tried a named entity recognition approach, using NTLK to tokenize, tag, and then chunk the Reddit comments into more recognizable entities. By counting all instances of PERSON chunk labels, we obtained an estimate of approximately 184 million references to people’s names across the twenty-three months. This result is the answer to the significantly different question of how many names of mentioned on Reddit, and only serves as an extreme upper bound for our original inquiry.

To narrow down the range of names that are linked to users actually identifying themselves, our second approach employed regular expressions and a couple heuristics. By searching first for occurrences of “My name is” and then for occurrences of “I” and taking the intersection of both searches, we obtained 21,264 results. This is a much more reasonable result, as our initial guess of 184 million represents 18.2% of all Reddit comments. At this point we
have three important observations. First, our regular expression search likely resulted in numerous false negatives, as it is conceivable that users could share their names without using the explicit phrase “My name is…” Second, the search also produced false positives, as several common results were movie quotes that clearly do not represent users’ names, such as Princess Bride’s “My name is Inigo Montoya.” or Fight Club’s “My name is Tyler Durden.” These Type II and Type I errors, respectively, clearly affect the result’s accuracy, but unfortunately we are unable to tell the extent of the error without already knowing the true answer.

Our final attempt at the name problem utilized both previous approaches and took the intersection of PERSON labels from chunking and the results of the above regular expressions. The resulting estimate of 18,865 users sharing their names eliminates some of the false positives from the second method, but otherwise suffers from the same drawbacks. We hazard a guess that the true number of sharing users is below 18,865, but ultimately have no way of knowing for sure.

We then moved on to finding instances of users sharing their addresses via Reddit. We used a near-identical methodology here, first looking for chains of CD-FACILITY-GPE chunk labels, then trying a “My address is…” heuristic, and finally a combination of both. Upon searching for the presence of consecutive CD, FACILITY, and GPE labels, we found only ten results. This was surprisingly low, as we expected far more addresses to be posted, especially when counting locations not linked to any single person. The primary reason for this vast underestimate is the NLTK chunker itself, which frequently mislabeled chunks when building up from POS tags. San Jose, California, for example, would be categorized as PERSON GPE instead of the correct GPE GPE due to its recognition of Jose as a name. However, the most popular alternatives to chunkers for identifying locations, known as gazetteers, often suffer from
the opposite problem. Gazetteers look up every tag in a large database of locations, but greedily classifies people or things as locations, such as classifying “Gary” as a city (as in Gary, Indiana) regardless of the context. With no good chunking methods available, we resorted to regular expressions, specifically looking for phrases like “My address is…” and “I live at/in/near…” to determine users’ addresses rather than generic locations. We found 293 such instances, which still seems on the low side. One very apparent downside to regular expression, chunking, and gazetter methods in this scenario is the ignorance of out-of-comment context. We are unable to identify any cases when users give out their address by simply answering a question instead of writing a full sentence themselves. This could be solved by examining parent and child comments, but the current structure of the Reddit comment corpus makes that extremely impractical.

Our last foray into user sharing looked into the presence of phone numbers across Reddit. The structure of phone numbers is much more fixed than that of addresses, which makes it considerably easier to search for. However, the effectiveness of named entity recognition—or at least NLTK’s flavor of it—is actually more constrained in this case. The only relevant entities are CD, which would unhelpfully group digits into larger numbers, and the specific parenthesis entities, which may not even be present in the phone number. We therefore chose a solely regular expression-based model, which allowed for flexibility with including parentheses, spaces, dashes, and the country code within the phone number. Our final result of 452,983 retrieved phone numbers is very high relative to the number of names we found earlier in this section. We expect few false positives for phone numbers, as users would have to either spell out the number in words or otherwise break the number up amongst other text to avoid being matched by the regular expression. False negatives are somewhat more likely, as long patterns
of numbers could potentially be wrongly identified as multiple phone numbers. Additionally, we have no way of verifying the authenticity of these phone numbers. It is worth noting that for certain subreddits such as RandomActsOfPizza (where users send each other deliveries of free pizza), handing out legitimate phone numbers may be encouraged or even required, which could contribute to our higher-than-expected phone count.

For the sake of comparison, Figure 11 below displays our best estimates for the number of names, addresses, and phone numbers shared throughout the comment corpus.

![Sharing Personal Information over Reddit](image)

**Figure 11: Shared Personal Information on Reddit**

**Social Media Patterns Analysis**

Our last section of NLP analysis looks into the characteristics of certain Reddit comments, specifically highly upvoted and overtly political comments. The consequences of
finding replicable patterns to funny, agreeable, or otherwise valuable comments are widespread, and are likely applicable to nearly every form of social media. Likewise, studying political discussions could have psychological insights into what is or is not convincing, or into the general direction that online debates tend to take. For both the high-scoring and political comment patterns, we use some machine learning in the form of Naïve Bayes classifiers to first extract important characteristics of a comment and then find out its overall predictive power. Note that these identifying features are selected by the classifier itself for having the most predictive power, and are not manually chosen by us.

In order to train our Naïve Bayes classifier, we first gathered the ten thousand highest-scoring comments on Reddit, along with ten thousand randomly selected low-scoring comments. We then extract a number of features from the comment, such as its NER chunking structure and the comment length. We now note that there is no surefire magic formula to writing high-scoring comments on Reddit, or similarly posting well-received text on any form of social media. In fact, often times there are factors related to the comment’s score that have nothing to do with the comment itself. Below are the results of the most informative features for identifying high-scoring comments.

<table>
<thead>
<tr>
<th>High-Scoring Comments: Most Informative Features</th>
<th>Feature</th>
<th>Feature Value</th>
<th>Likely Label</th>
<th>Likelihood Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>subreddit DotA2</td>
<td>&lt;1000</td>
<td>55.0 : 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subreddit nfl</td>
<td>&lt;1000</td>
<td>47.0 : 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subreddit hockey</td>
<td>&lt;1000</td>
<td>39.8 : 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subreddit nba</td>
<td>&lt;1000</td>
<td>28.7 : 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subreddit trees</td>
<td>&lt;1000</td>
<td>25.8 : 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subreddit atheism</td>
<td>&lt;1000</td>
<td>23.7 : 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subreddit pokemon</td>
<td>&lt;1000</td>
<td>23.7 : 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>user AutoModerator</td>
<td>&lt;1000</td>
<td>22.3 : 1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subreddit Android</td>
<td>&lt;1000</td>
<td>19.7 : 1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2 presents the feature values that are most strongly linked to a specific label. For high-scoring comments, we have two labels: comments that have a score of below one thousand (<1000) and everything else (>1000). For example, if a comment is made in the DotA2 subreddit, it is fifty-five times more likely than average to have under one thousand points. Similarly, the AutoModerator user (a bot) is disproportionately likely to have low-scoring comments. Without even looking at the comment structure, we learn that several sports and technology-based subreddits are either unusually begrudging with their upvotes or simply do not have as many “good” comments worth upvoting. We additionally learn that successfully commenting on Reddit (and likely other social media) shares a tenet with real estate: location, location, location. Commenting on certain subreddits, especially the ones listed in Table 2, are more likely to either get buried alongside countless other users or downvoted and hidden from view.

We now look at the structure of the comments themselves for further insight into what makes causes some comments to be so highly upvoted. Using tokenization, POS-tagging, and name entity recognition chunking yet again, we extract several levels of the chunking tree and analyze those features with another Naïve Bayes classifier.
We note that while comment structure has less predictive power than knowing the subreddit or user, it is a useful feature nonetheless. Below are examples of comments that match the subtree or leaf labels for all feature values in Table 3:

- ('NN', '.') - Nice!
- ('NNP', 'NNP', '.', 'NNP', 'NNP') - Jedi Sprinkles.
- ('S', 'LOCATION') - The Plaza.
- ('NN', '.', 'NN') - *Block-gasm!* 
- ('CD',) - 5
- ('DT', 'NN', '.') - No prob.
- ('JJ', 'NN') - super advertising
- ('RB', '.') - Hardly.
- (',', ')') - :
- ('S', 'GPE', 'PERSON', 'ORGANIZATION', 'ORGANIZATION') - Um...what is Defense Grid? God, I feel like Microsoft is mocking us.
As another example, our model finds that without knowing anything more, the comment “No prob.”, with the structure ('DT', 'NN', '.'), is 5.7 times more likely than average to have a score over one thousand. However, none of the linguistic patterns in Table 3 especially stands out, which is revealing in and of itself. It is understandable why certain patterns are disproportionately low-scoring; simple adverbs (RB) or emoji ( :) ) generally do not contribute much to online discussions, and thus would be far less likely to reach one thousand points. However, a sentence ending in a location (S LOCATION) is unexpectedly a positive feature value, as is a determiner followed by a noun (DT NN). We conclude here that while these linguistic structures may indeed have some je n'ai quoi, we are unable to identify why exactly it is important.

Our last remarks regarding the features of high-scoring comments deal with the predictive power of our classifier. Using the same trained classifier as before, we found an accuracy of 58.1% on a disjoint testing set when guessing whether a comment would have a score below or above one thousand. This accuracy, although admittedly low, is statistically significantly better than a random guess. When we take into account additional features, including the subreddit and user as shown in Table 2, we reach an accuracy of 60.0%. This accuracy could be further improved by taking more features into account, but at the cost of exponentially increasing the training set size due to the curse of dimensionality.

We again trained a Naïve Bayes classifier for identifying the definitive features of both political and apolitical comments. However, there is no easy apparent way to gather political comments to construct a training set. As a fairly effective heuristic, we chose to define all meaningful comments (those with twenty or more points) in the politics subreddit as political.
Similarly, we gathered comments from the science, askscience, and AskHistorians subreddits to create our apolitical training and testing set. We admit that this solution is not perfect, as not all comments in the politics subreddit are necessarily political, nor is the content of three specific subreddits representative of all apolitical comments on Reddit. After collecting ten thousand comments for each label, we extracted similar features as with the high-scoring comments and tested their effects on classification.

**Table 4: Most Informative Political Structural Features**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature Value</th>
<th>Likely Label</th>
<th>Likelihood Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains 'Republican'</td>
<td>True</td>
<td>Political</td>
<td>25.5 : 1.0</td>
</tr>
<tr>
<td>Contains 'Democrat'</td>
<td>True</td>
<td>Political</td>
<td>24.4 : 1.0</td>
</tr>
<tr>
<td>Subtree Label</td>
<td>('RB', 'VBN', 'NN')</td>
<td>Apolitical</td>
<td>7.5 : 1.0</td>
</tr>
<tr>
<td>Contains 'Trump'</td>
<td>True</td>
<td>Political</td>
<td>5.7 : 1.0</td>
</tr>
<tr>
<td>Short Comment</td>
<td>True</td>
<td>Apolitical</td>
<td>5.1 : 1.0</td>
</tr>
<tr>
<td>Leaf Label</td>
<td>('S', 'GPE', 'GSP')</td>
<td>Political</td>
<td>5.1 : 1.0</td>
</tr>
<tr>
<td></td>
<td>('S', 'ORGANIZATION', 'ORGANIZATION', 'ORGANIZATION', 'ORGANIZATION', 'ORGANIZATION', 'ORGANIZATION')</td>
<td>Political</td>
<td>5.1 : 1.0</td>
</tr>
<tr>
<td>Leaf Label</td>
<td>'ORGANIZATION', 'ORGANIZATION', 'ORGANIZATION'</td>
<td>Political</td>
<td>5.1 : 1.0</td>
</tr>
<tr>
<td>Subtree Label</td>
<td>('NNP', 'PRP', '.')</td>
<td>Apolitical</td>
<td>4.6 : 1.0</td>
</tr>
<tr>
<td>Leaf Label</td>
<td>('S', 'ORGANIZATION', 'GSP')</td>
<td>Political</td>
<td>4.4 : 1.0</td>
</tr>
<tr>
<td>Leaf Label</td>
<td>('S', 'PERSON', 'PERSON', 'GPE')</td>
<td>Political</td>
<td>4.2 : 1.0</td>
</tr>
</tbody>
</table>

The results of this classifier contains both intuitive and some unintuitive results. Comments that contain the words “Democrat,” “Republican”, or “Trump”—even back in 2013—are all significantly more likely than average to be political. However, short comments are more likely not to be political, for reasons that are not entirely clear. It is possible that our threshold of twenty points for political comments removed many short comments, but perhaps science comments are simply more concise. We now provide examples for comments that match the language structure of the six relevant comments in Table 4:
• ('RB', 'VBN', 'NN') - nicely done sir

• ('S', 'GPE', 'GSP') - I thought China depends on US to buy their exports.

• ('S', 'ORGANIZATION', 'ORGANIZATION', 'ORGANIZATION', 'ORGANIZATION', 'ORGANIZATION', 'ORGANIZATION') - If you've got any T-Mobile Sim card you've got access to T-mobile LTE.

• ('NNP', 'PRP', '.') - Thank you!

• ('S', 'ORGANIZATION', 'GSP') - Everything is VERY illegal in the US.

• ('S', 'PERSON', 'PERSON', 'GPE') - Try Deadpool: Pulp if you can find it. It's kind of like Marvel Noir but not set in that universe

Like with the high-scoring comments, we are unable to induct much meaning from the linguistics of the most informative grammatical structures. The predictive power for the political comment classifier is 63.94%, slightly higher than that of the score-based model. Certain feature values, like containing “Democrat” or “Republican,” appear to be dead giveaways, and are almost never present in apolitical comments unless in a historical context. The power of these two features alone, 25.5 : 1 and 24.4 : 1, is likely what gives this model the predictive edge over the former.

For both Naïve Bayes classifiers, we do not immediately come away with an understanding of why certain language patterns are more closely linked to either high-scoring or more political comments.
6 Opportunities for Further Study

For each of the three main areas of NLP analysis that we explored, there are numerous opportunities to explore more deeply and refine approximate solutions. We divide these opportunities by category in accordance with our analysis in the previous chapter.

For Reddit phenomena, there are dozens or even hundreds of other text-based memes to be analyzed in a similar manner. Studying copypastas like the Navy Seal one over longer periods of time could yield interesting results, and help determine the average lifespan of a meme or if some memes never really die out. Further research could look at other copypastas or integrate image recognition to begin looking at the mountain of image-based memes. Perhaps the biggest source of new information could come from studying the content of Reddit posts, rather than just the comments. Reddit users must open a post in order to view the attached link, text, or any associated comments. Due to the constant influx of new Reddit posts, there is likely a massive syndrome of judging books by their cover, thereby causing potentially high-scoring comments to be ignored. In that vein, one could look at whether reposted posts are more likely to contain reposted comments, and if there is any rhyme (if not reason) to the reposted posts.

Future work into sharing and privacy concerns would quickly run into the same problem of identifying what information is genuinely personal. It is possible to look for less structured information, such as relationship status, career, hobbies, and shopping habits, which Reddit considers less directly personal and thus does not ban or remove. Regarding Reddit moderation, one could research whether badly moderated subreddits (especially new and very large subreddits) are more likely to contain personal information, which could have many security and social engineering implications.
Of the three main categories covered in this thesis, the area of identifying and predicting traits of comments likely has the most potential. With either a substantial time investment or a more accurate heuristic, there could be improvements upon the predictive models for political comments. It would also be interesting to see whether the number of political comments increases over time, or some similar quantification of political polarization (or more surprisingly, the lack thereof). We could also create non-binary labels for our model, such as predicting comments that have scores of less than zero points, zero to ten points, ten to one hundred points, and above one hundred points, instead of just greater or less than one thousand. There are also many other patterns beyond score and politicalness to look for in the first place, such as controversiality, Reddit gilding, and the level of responses or interactions by other users—and each with its own classifier. With more time, we would look to add features to these classifiers, beyond the presence of some key phrases and its general chunking structure. Checking for the number of nouns, verbs or adjectives, for example, could bring be a powerful predictor on its own, while the time of posting could have a huge impact on a comment’s score. The biggest open question we were unable to answer is the reasoning behind the chunking structures indicative of high-scoring or political comments. Further analysis of those patterns and the matching comments could result in several new insights, and bring us one step closer to the magic formula for commenting that we alluded to in the introduction.
Appendix A

main/Utils.py

```python
import nltk

class Utils:
    # Constants
    CORPUS_PATHNAME = 'F:/Reddit Comments Corpus'
    USER_DATA_PATHNAME = './UserData'
    _2013_MONTHS = [x for x in range(7,13)]
    _2014_MONTHS = [x for x in range(1,13)]
    _2015_MONTHS = [x for x in range(1,6)]
    ALL_FILES = {2013 : _2013_MONTHS, 2014 : _2014_MONTHS, 2015 :
    _2015_MONTHS}

    @staticmethod
    # Convenience method for accessing the locally stored comment corpus
    def constructFilePath(year, month):
        year = str(year)
        if month < 10: # Adjust the month section of the path string as needed
            month = '0' + str(month)
        else:
            month = str(month)

        filePath = Utils.CORPUS_PATHNAME + '/' + year + '/RC_' + year + '-' +
        month + '.json'
        return filePath

    # Use closures to easily iterate over all months
    @staticmethod
    def forAllMonths(f):
        for year in Utils.ALL_FILES:
            for month in Utils.ALL_FILES[year]:
                with open(Utils.constructFilePath(year, month)) as file:
                    f(year, month, file)

    # Return a list of all subtree (non-leaf) chunk labels
    @staticmethod
    def getSubtreeChunkLabels(comment):
        tokens = nltk.word_tokenize(comment)
        tags = nltk.pos_tag(tokens)
        chunks = nltk.ne_chunk(tags)

        labelList = []
        for subtree in chunks.subtrees():
```
for leaf in subtree.leaves():
    labelList.append(leaf[1])

return tuple(labelList)

# Return a list of all leaf-label POS labels
@staticmethod
def getLeafChunkLabels(comment):
    tokens = nltk.word_tokenize(comment)
    tags = nltk.pos_tag(tokens)
    chunks = nltk.ne_chunk(tags)

    labelList = []
    for subtree in chunks.subtrees():
        labelList.append(subtree.label())

    return tuple(labelList)

main/SummaryStatistics.py

'''
Created on Feb 12, 2017
@author: Peter
'''

import json
import numpy as np
import matplotlib.pyplot as plt
from functools import reduce
from main import Utils

# Find the monthly totals for the number of comments and the sum of comment scores,
# then write to file
def storeTotalCommentCounts():
    def storeCommentCounts(year, month, file):
        userCount = 0
        totalPoints = 0

        for line in file:
            commentDict = json.loads(line)
            userCount += 1
            totalPoints += commentDict['score']

        pathName = Utils.USER_DATA_PATHNAME + '/Comment Counts.txt'

        # Save the month/year,
        f = open(pathName, 'w')
        f.write(year + '-' + month + '
')
f.write(userCount + 'n')
f.write(totalPoints + 'n')
f.write(totalPoints/userCount + 'n')
Utils(forAllMonths(storeCommentCounts)

# Create a dictionary of all user : number of comments) entries, then write to file
def storeAllUserCommentCounts():
    def storeUserCommentCounts(year, month, file):
        userDict = {}
        for line in file:
            commentDict = json.loads(line)
            user = commentDict['author']

            if user in userDict:
                userDict[user] += 1
            else:
                userDict[user] = 1

        pathname = '../UserData/' + str(year) + '-' + str(month) + '.json'
        outputFile = open(pathname, 'w+')
        outputFile.write(json.dumps(userDict))
        outputFile.close()

    Utils(forAllMonths(storeUserCommentCounts)

# Create three bar charts for the number of comments, sum of all comment scores, and average score per comment over the time interval
def showTotalUserAndCommentCounts():
    pathName = 'Comment and Point Counts.txt'
    commentList = []
    pointList = []
    averagePointsList = []

    with open(pathName) as file:
        lineCount = 0
        for line in file:
            if lineCount % 3 == 1:
                commentList.append(int(line))
            elif lineCount % 3 == 2:
                pointList.append(int(line))
            lineCount += 1

        # Calculate monthly average scores by iterating over combined list
        for pointCount, userCount in zip(pointList, commentList):
            averagePointsList.append(pointCount/userCount)

        print(sum(commentList))

# Create an evenly spaced interval using the desired months
y_pos = np.arange(len(dates))

# Bar graph of monthly comment totals
plt.figure(1)  # Define first figure to be displayed
plt.bar(y_pos, commentList, align='center')  # Create the actual bar graph
plt.xticks(y_pos, dates, rotation='vertical')  # Define labels per x value on the x axis
plt.title('Total Comment Counts')
plt.xlabel('Month')
plt.ylabel('Number of Users')
plt.tight_layout()  # Ensure that we have enough space for the graph and all text

# Bar graph of monthly comment score totals
plt.figure(2)
plt.bar(y_pos, pointList, align='center')
plt.xticks(y_pos, dates, rotation='vertical')
plt.title('Total Point Counts')
plt.xlabel('Month')
plt.ylabel('Number of Points')
plt.tight_layout()

# Bar graph of average comment scores
plt.figure(3)
plt.bar(y_pos, averagePointsList, align='center')
plt.xticks(y_pos, dates, rotation='vertical')
plt.title('Average Comment Scores')
plt.xlabel('Month')
plt.ylabel('Average Number of Points')
plt.tight_layout()

plt.show()  # Display all figures

# Fine the number of comments each user makes, then visualize in a histogram

def showUniqueMonthlyCommentCounts():
    pointCountList = []
    for year in Utils.ALL_FILES:
        for month in Utils.ALL_FILES[year]:
            pathName = Utils.USER_DATA_PATHNAME + './' + str(year) + '-' + str(month) + '.json'
            with open(pathName) as file:
                line = file.readline()
                file.close()
                testDict = json.loads(line)

                pointCountList += list(testDict.values())

    # Create a logarithmic-scale histogram to better display all data
    plt.hist(pointCountList, bins=np.logspace(0, 4.5, 50))
    plt.gca().set_xscale('log')
    plt.yscale('log', nonposy='clip')
plt.title('User Activeness Distribution')
plt.xlabel('Number of Comments')
plt.ylabel('Number of Users')
plt.show()

print('Median comments per user: ', np.median(np.array(pointCountList)))
print('Mean comments per user: ', reduce(lambda x, y: x + y, pointCountList) / len(pointCountList))

main/NLP/RedditPhenomena.py

'''Created on Mar 9, 2017
@author: Peter
'''

import json
import re
import collections
import numpy as np
import matplotlib.pyplot as plt

from main import Utils

#create a new copy of Reddits
#Find all instances of the Navy Seal copypasta
def findMemeFrequency():
    pattern = r'What the \(fuck\) did you just fucking say about me'
    outputPath = './Memes.json'
    outputFile = open(outputPath, 'w')

    for year in Utils.ALL_FILES:
        for month in Utils.ALL_FILES[year]:
            with open(Utils.constructFilePath(year, month)) as file:
                for line in file:
                    commentDict = json.loads(line)
                    comment = commentDict['body']
                    meme = re.findall(pattern, comment)

                    if meme:
                        commentDict.pop('body')
                        outputFile.write(json.dumps(commentDict) + '
')

#Find variations of the Navy Seal copypasta
def findCopypastaVariants():
    copypastaVariants = {}
    pattern = r'What the \(fuck\) did you just fucking say about me'
    pathname = './CopypastaVariants.json'

    for year in Utils.ALL_FILES:
        for month in Utils.ALL_FILES[year]:
            with open(Utils.constructFilePath(year, month)) as file:
for line in file:
    commentDict = json.loads(line)
    comment = commentDict['body']
    meme = re.findall(pattern, comment)

    if meme:
        if comment in copypastaVariants:
            copypastaVariants[comment] += 1
        else:
            copypastaVariants[comment] = 1

outputFile = open(pathname, 'w')
outputFile.write(json.dumps(copypastaVariants) + '\n')

# Track all occurrences of reposted comments
def findReposts():
    for year in Utils.ALL_FILES:
        for month in Utils.ALL_FILES[year]:
            with open(Utils.constructFilePath(year, month)) as file:
                commentQueue = collections.deque(maxlen = 10000)
                repostList = []
                for line in file:
                    commentDict = json.loads(line)
                    comment = commentDict['body']

                    if len(comment) > 20:
                        if comment in commentQueue:
                            repostList.append(comment)
                        elif '[deleted]' not in comment:
                            commentQueue.append(comment)

                print(year, month, len(repostList))

# Display graph of Navy Seal copypasta occurrences by month
# Numbers are hardcoded in from results
def showNavySealFrequency():
    memeList = [214, 230, 117, 142, 148, 221, 212, 162, 166, 178, 212, 221,
                273, 305, 269, 271, 249, 248, 275, 224, 249, 276, 241]


    y_pos = np.arange(len(dates))
    plt.bar(y_pos, memeList, align='center', alpha=0.5)
    plt.xticks(y_pos, dates)
    plt.yticks(y_pos, dates, rotation = 'vertical')
    plt.ylabel('Number of Instances')
    plt.title('Occurrences of the Navy Seal (Gorilla Warfare) copypasta')
```python
plt.tight_layout()  # Ensure that we have enough space for the graph and all text
plt.show()

# Display graph of Navy Seal copypasta variants by theme
# Numbers are hardcoded in from results
def showNavySealVariants():
    variantList = [5114, 1331, 199, 184, 175, 140, 5569]
    variantNames = ('Original', 'Shortened', 'Social Justice', 'Star Trek', 'WWE', 'Economics', 'Other')

    y_pos = np.arange(len(variantNames))
    plt.bar(y_pos, variantList, align='center', alpha=0.5)
    plt.xticks(y_pos, variantNames)
    plt.xticks(y_pos, variantNames, rotation='vertical')
    plt.xlabel('Variant Name or Theme')
    plt.ylabel('Frequency')
    plt.title('Reddit Navy Seal Copypasta Variants')
    plt.tight_layout()  # Ensure that we have enough space for the graph and all text
    plt.show()

# Display graph of all reposted comments by month
# Numbers are hardcoded in from results
def showReposts():
    repostList = [80215, 75904, 77481, 167212, 166177, 169232, 175839, 157675, 185685, 180263, 201595, 149913, 173088, 151904, 164213, 210970, 170100, 227799, 177997, 198683, 221371, 206748]
    numCommentsList = [34922133, 34766579, 31990369, 35940040, 37396497, 39810216, 42420655, 38703362, 42459956, 42440735, 42514094, 46868899, 46990813, 44992201, 47497520, 46118074, 48807699, 53851542, 48342747, 54564441, 55005780, 54504410]

    normalizedList = []
    for month, num in zip(repostList, numCommentsList):
        normalizedList.append(month * 34922133/num)

    print(normalizedList)
    y_pos = np.arange(len(dates))

    plt.bar(y_pos, normalizedList, align='center', alpha=0.5)
```
```python
plt.xticks(y_pos, dates)
plt.xticks(y_pos, dates, rotation = 'vertical')
plt.ylabel('Number of Reposts')
plt.title('All Reddit Reposts, Adjusted to July 2013')
plt.tight_layout() #Ensure that we have enough space for the graph and all text
plt.show()

main/NLP/RedditSharing.py

'''
Created on Mar 9, 2017
@author: Peter
'''

import json
import re
import nltk
import numpy as np
import matplotlib.pyplot as plt
from main import Utils

#Find all occurrences of personal names using regular expressions
def findNamesRegex():
    for year in Utils.ALL_FILES:
        for month in Utils.ALL_FILES[year]:
            nameCount = 0
            with open(Utils.constructFilePath(year, month)) as file:
                for line in file:
                    commentDict = json.loads(line)
                    comment = commentDict['body']
                    regexResult = re.findall('[Mm]y name is [A-Z][a-z]+ [A-Z][a-z]+', comment)
                    result2 = re.findall('I ', comment)
                    if regexResult and result2:
                        nameCount += 1
            print(year, month, nameCount)

#Find all occurrences of personal names using named entity recognition
def findNamesNER():
    for year in Utils.ALL_FILES:
        for month in Utils.ALL_FILES[year]:
            nameCount = 0
            with open(Utils.constructFilePath(year, month)) as file:
                for _ in range(10000):
                    line = file.readline()
                    commentDict = json.loads(line)
                    comment = commentDict['body']
                    tokens = nltk.word_tokenize(comment)
                    tags = nltk.pos_tag(tokens)
                    chunks = nltk.ne_chunk(tags)
```

for subtree in chunks.subtrees(filter=lambda x: x.label() == 'PERSON'):
    if subtree.leaves():
        nameCount += 1
    break
print(year, month, nameCount)

# Find all occurrences of personal addresses using regexes and NER
def findAddresses():
    chunkAddressCount = 0
    regexAddressCount = 0
    combinedAddressCount = 0
    for year in Utils.ALL_FILES:
        for month in Utils.ALL_FILES[year]:
            pattern1 = r'My address is'
            pattern2 = r'I live (at|in|on|near)'
            with open(Utils.constructFilePath(year, month)) as file:
                for _ in range(10000):
                    line = file.readline()
                    commentDict = json.loads(line)
                    comment = commentDict['body']
                    tokens = nltk.word_tokenize(comment)
                    tags = nltk.pos_tag(tokens)
                    chunks = nltk.ne_chunk(tags)

                    regexResult1 = re.findall(pattern1, comment)
                    regexResult2 = re.findall(pattern2, comment)

                    if regexResult1 or regexResult2:
                        regexAddressCount += 1

            for subtree in chunks.subtrees(filter=lambda t:
                                            t.label() == 'FACILITY' or
                                            t.label() == 'GPE' or
                                            t.label() == 'LOCATION'):
                if len(subtree.leaves()) > 3:
                    chunkAddressCount += 1

            if regexResult1 or regexResult2:
                combinedAddressCount += 1

        print('Total Chunking', chunkAddressCount)
        print('Total Regex', regexAddressCount)
        print('Total Combined', combinedAddressCount)

    # Find all occurrences of personal phone numbers using regular expressions
    def findPhoneNumbers():
        pattern = re.compile(r'(?:(?:\(?[\d]{3}\)?|\d{0,3})\d{3}[-\d]{4}).*')
        for year in Utils.ALL_FILES:
            for month in Utils.ALL_FILES[year]:
                phoneCount = 0
with open(Utils.constructFilePath(year, month)) as file:
    for line in file:
        commentDict = json.loads(line)
        comment = commentDict['body']
        phoneNumbers = re.findall(pattern, comment)

        if phoneNumbers:
            phoneCount += 1
        print(year, month, phoneCount)

#Display all results of sharing tendencies
#Data is hardcoded in from results

def showAllResults():
    numNames = 18865
    numAddresses = 293
    numPhoneNumbers = 452983

    objects = ('Names', 'Addresses', 'Phone Numbers')

    y_pos = np.arange(len(objects))
    sharingList = [numNames, numAddresses, numPhoneNumbers]

    plt.bar(y_pos, sharingList, align='center', alpha=0.5)
    plt.xticks(y_pos, objects)
    plt.yscale('log')
    plt.ylabel('Number of Instances')
    plt.title('Sharing Personal Information over Reddit')
    plt.show()

showAllResults()
POLITICAL_COMMENTS_PATH = './PoliticalComments.txt'
NONPOLITICAL_COMMENTS_PATH = './NonPoliticalComments.txt'
TRAINING_CAPACITY = 10000
NUM_MONTHS = 23
LOW_COMMENT_THRESHOLD = 500

# Locally store high-scoring comments
def storeHighScoringComments():
    commentList = []
    lineCount = 0
    for year in Utils.ALL_FILES:
        for month in Utils.ALL_FILES[year]:
            with open(Utils.constructFilePath(year, month)) as file:
                for line in file:
                    lineCount += 1
                    try:
                        commentDict = json.loads(line)
                    except (JSONDecodeError, TypeError):
                        pass
                    commentEntry = (commentDict['score'],
                                    commentDict['author'],
                                    commentDict['subreddit'],
                                    commentDict['body'])
                    commentList.append(commentEntry)
                    if (lineCount % (5 * TRAINING_CAPACITY) is 0):
                        commentList = heapq.nlargest(TRAINING_CAPACITY,
                                                     commentList,
                                                     lambda x : x[0])
            commentList = heapq.nlargest(TRAINING_CAPACITY, commentList,
                                         lambda x : x[0])
            outputFile = open(HIGH_SCORING_COMMENTS_PATH, 'w')
            for comment in commentList:
                try:
                    outputFile.write(str(comment) + '\n')
                except (UnicodeEncodeError):
                    pass

# Locally store low-scoring comments
def storeLowScoringComments():
    for year in Utils.ALL_FILES:
        for month in Utils.ALL_FILES[year]:
            with open(Utils.constructFilePath(year, month)) as file:
                commentList = []
                commentCount = 0
                while commentCount < TRAINING_CAPACITY / NUM_MONTHS:
                    line = file.readline()
                    commentDict = json.loads(line)
                    if commentDict['score'] < LOW_COMMENT_THRESHOLD:
                        commentEntry = (commentDict['score'],
                                        commentDict['author'],
                                        commentDict['subreddit'],
                                        commentDict['body'])
                        commentList.append(commentEntry)
                        commentCount += 1
outputFile = open(LOW_SCORING_COMMENTS_PATH, 'a')
for comment in commentList:
    try:
        outputFile.write(str(comment) + '\n')
    except (UnicodeEncodeError):
        pass

# Use a Naive Bayes classifier to predict high-scoring comments

def predictHighScoringComments():
    lowScoringCommentsFile = open(LOW_SCORING_COMMENTS_PATH)
    highScoringCommentsFile = open(HIGH_SCORING_COMMENTS_PATH)
    commentsList = []
    for lowLine, highLine in zip(lowScoringCommentsFile, highScoringCommentsFile):
        try:
            commentsList.append((literal_eval(lowLine), '<1000'))
            commentsList.append((literal_eval(highLine), '>1000'))
        except ValueError:
            continue
    random.shuffle(commentsList)

    featureSet = [(commentFeatures(commentEntry), label) for (commentEntry, label) in commentsList]
    trainSet, testSet = featureSet[int(TRAINING_CAPACITY / 2):], featureSet[:int(TRAINING_CAPACITY / 2)]
    classifier = nltk.classify.NaiveBayesClassifier.train(trainSet)
    print(nltk.classify.accuracy(classifier, testSet))
    classifier.show_most_informative_features()

# Extract classifier features for high-scoring comments

def commentFeatures(commentEntry):
    score, user, subreddit, comment = commentEntry
    subtreeChunkLabels = Utils.getSubtreeChunkLabels(comment)
    leafChunkLabels = Utils.getLeafChunkLabels(comment)
    shortComment = len(comment) <= 20
    medComment = (len(comment) > 20) and (len(comment) < 50)
    longComment = len(comment) > 50

    features = {
        'subtreeChunkLabel': subtreeChunkLabels,
        'leafChunkLabel': leafChunkLabels,
        'shortComment': shortComment,
        'medComment': medComment,
        'longComment': longComment
    }
    return features

# Locally store political comments

def storePoliticalComments():
    commentList = []
    for year in Utils.ALL_FILES:
for month in Utils.ALL_FILES[year]:
    with open(Utils.constructFilePath(year, month)) as file:
        for line in file:
            try:
                commentDict = json.loads(line)
            except(JSONDecodeError, TypeError):
                pass

            if commentDict['subreddit'] == 'politics' and commentDict['score'] > 10:
                commentEntry = (commentDict['score'],
                                commentDict['author'], commentDict['subreddit'], commentDict['body'])
                commentList.append(commentEntry)

outputFile = open(POLITICAL_COMMENTS_PATH, 'w')
for comment in commentList:
    try:
        outputFile.write(str(comment) + '
')
    except(UnicodeEncodeError):
        pass

#Locally store apolitical comments
def storeNonPoliticalComments():
    for year in Utils.ALL_FILES:
        for month in Utils.ALL_FILES[year]:
            with open(Utils.constructFilePath(year, month)) as file:
                commentList = []
                commentCount = 0
                while commentCount < TRAINING_CAPACITY / NUM_MONTHS:
                    line = file.readline()
                    commentDict = json.loads(line)

                    if ((commentDict['subreddit'] == 'AskHistorians') or (commentDict['subreddit'] == 'askscience') or (commentDict['subreddit'] == 'science')):
                        commentEntry = (commentDict['score'],
                                        commentDict['author'], commentDict['subreddit'], commentDict['body'])
                        commentList.append(commentEntry)
                        commentCount += 1

                outputFile = open(NONPOLITICAL_COMMENTS_PATH, 'a')
                for comment in commentList:
                    try:
                        outputFile.write(str(comment) + '
')
                    except(UnicodeEncodeError):
                        pass

#Use a Naive Bayes classifier to predict political comments
def predictPoliticalComments():
    nonPoliticalCommentsFile = open(NONPOLITICAL_COMMENTS_PATH)
    politicalCommentsFile = open(POLITICAL_COMMENTS_PATH)
    commentsList = []
for nonPolLine, polLine in zip(nonPoliticalCommentsFile, politicalCommentsFile):
    commentsList.append((literal_eval(nonPolLine), 'Nonpolitical'))
    commentsList.append((literal_eval(polLine), 'Political'))

random.shuffle(commentsList)

featureSet = [(politicalCommentFeatures(commentEntry), label) for (commentEntry, label) in commentsList]
trainSet, testSet = featureSet[:int(TRAINING_CAPACITY / 2)], featureSet[int(TRAINING_CAPACITY / 2):]

classifier = nltk.classify.NaiveBayesClassifier.train(trainSet)
print(nltk.classify.accuracy(classifier, testSet))
classifier.show_most_informative_features()

# Extract classifier features for political comments

def politicalCommentFeatures(commentEntry):
    score, user, subreddit, comment = commentEntry
    subtreeChunkLabels = Utils.getSubtreeChunkLabels(comment)
    leafChunkLabels = Utils.getLeafChunkLabels(comment)
    shortComment = len(comment) <= 20
    medComment = (len(comment) > 20) and (len(comment) < 50)
    longComment = len(comment) > 50
    democrat = 'democrat' in comment.lower()
    republican = 'republican' in comment.lower()
    trump = 'trump' in comment.lower()

    features = {
        'subtreeChunkLabel': subtreeChunkLabels,
        'leafChunkLabel': leafChunkLabels,
        'shortComment': shortComment,
        'medComment': medComment,
        'longComment': longComment,
        'democrat': democrat,
        'republican': republican,
        'trump': trump
    }

    return features
Bibliography


Peter Mason

EDUCATION
Pennsylvania State University, University Park, PA
B.S., Computer Science, Mathematics minor
Schreyer Honors College member

Spring 2017 Graduation

RELATED EXPERIENCE
J.P. Morgan Chase, Newark, DE
Technology Analyst Intern
Summer 2016
- Developed financial reconciliation service using Spring and AngularJS
- Introduced automated regression testing to internal web application

IBM Kenexa Lab, Wayne, PA
Software Engineering Intern
Summer 2015
- Developed and debugged code for large web application
- Automated testing for web application and IBM’s public SOAP and REST APIs

Pennsylvania State University Applied Research Laboratory, University Park, PA
Software Development Intern
Summer 2014 - Present
- Developed, tested, and debugged large-scale data visualization software tool
- Worked with a team to parse and visualize Bitcoin blockchain data

COMPUTER AND PROGRAMMING SKILLS
Advanced Java experience (5 years)
- Developed data visualization tool for time-dependent data sets while at ARL

Intermediate Python and Groovy (3 years)
- Wrote hackathon apps and value-at-risk Monte Carlo simulation in Python

Intermediate MySQL and OracleDB experience (1 year)
- Created backend for storing history of financial reconciliation “runs” using Oracle DB while at JPMC, saving users from repeating time-consuming tasks

ACCOMPLISHMENTS
AEC 3.0 Hackathon, Best Virtual Reality Hack winner
February 2016
- Developed a building and texture simulation for the HTC Vive using Unity

Code for Good 2015 hackathon participant
October 2015
- Developed website and server to address inventory and information needs for Goodwill Delaware

HackMIT 2014 participant
October 2014
- Developed Bitcoin exploration tool to track flow of purchases (GitHub repo at https://github.com/pem5134/HackMIT)

Eagle Scout
Spring 2013
- Led team to construct and install twenty-one bluebird houses in local nature preserve

ACTIVITIES
Theory of Computation Learning Assistant
Spring 2016 - Present
- Assisted students and graded papers regarding regular and context-free languages, Turing machines, NP-completeness, and other topics

Member, Association for Computing Machinery
Fall 2013 - Present
- Organized and attended presentations on a variety of computer science topics