A METHOD FOR SEGMENTING TOPICAL TWITTER HASHTAGS

A Thesis
Presented to the
Faculty of
San Diego State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Arts
in
Linguistics

by
Brandon James Devine
Spring 2014
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Brandon James Devine:

A Method of Segmenting Topical Twitter Hashtags

Robert P. Malouf, Chair
Department of Linguistics and Asian/Middle Eastern Languages

Jean Mark Gawron
Department of Linguistics and Asian/Middle Eastern Languages

André Skupin
Department of Geography

3-20-2014
Approval Date
DEDICATION

For Wendy
“When I use a word,” Humpty Dumpty said, in a rather scornful tone, “it means just what I choose it to mean – neither more nor less.”
“The question is,” said Alice, “whether you can make words mean so many different things.”
“The question is,” said Humpty Dumpty, “which is to be master – that’s all.”

-- Lewis Carroll, *Through the Looking-Glass and What Alice Found There*

One describes a tale best by telling the tale. You see? The way one describes a story, to oneself or the world, is by telling the story. It is a balancing act and it is a dream. The more accurate the map, the more it resembles the territory. The most accurate map possible would be the territory, and thus would be perfectly accurate and perfectly useless. The tale is the map that is the territory.

-- Neil Gaiman, *Fragile Things*
ABSTRACT OF THE THESIS

A Method of Segmenting Topical Twitter Hashtags
by
Brandon James Devine
Master of Arts in Linguistics
San Diego State University, 2014

Hashtags are a feature of tweets sometimes utilized in identifying discourse topic and/or user sentiment, presented in the form #thisisahashtag. This method of word concatenation, while a logical response to the 140-character limit per tweet enforced by Twitter, does present certain issues in the context of applying machine learning techniques to derive a tweet’s topic or sentiment: since the training data for a tweet are limited, one naturally wishes to utilize said data to the fullest extent possible, but one well-known method for segmenting strings does not necessarily work in the context of hashtags. This potential underperformance stems from a reliance on a static training corpus when generating n-gram probabilities; the ephemeral nature of hashtags that respond to current events and trends indicates instead a need for a dynamically updated training corpus.

This thesis proposes a modification of said method that retains its probabilistic power while allowing slang and other neologisms typically found in hashtags to bubble up through the training data at a rate that permits proper segmentation on terms that would otherwise not be recognized. I begin with a review of tweet structure and the algorithm that is to be modified and then discuss certain operational and methodological issues inherent in working with Twitter data. I then describe my algorithm and the techniques utilized in avoiding said issues. I conclude with a discussion of the new algorithm’s efficacy and ways in which it might be improved.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>x</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>xi</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 An Overview of Twitter Mechanics</td>
<td>1</td>
</tr>
<tr>
<td>1.2 General Motivation for This Work</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Related Work</td>
<td>7</td>
</tr>
<tr>
<td>1.3.1 The Web as Corpus</td>
<td>7</td>
</tr>
<tr>
<td>1.3.2 Twitter as a Medium of Communication</td>
<td>8</td>
</tr>
<tr>
<td>1.3.3 Existing Segmentation Techniques</td>
<td>8</td>
</tr>
<tr>
<td>2 THE PARAMETERS OF TWITTER</td>
<td>11</td>
</tr>
<tr>
<td>2.1 A User Typology</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Tweet Structure</td>
<td>13</td>
</tr>
<tr>
<td>2.2.1 A Sample Tweet</td>
<td>13</td>
</tr>
<tr>
<td>2.2.2 Hashtags</td>
<td>14</td>
</tr>
<tr>
<td>3 A METHOD OF IMPROVED HASHTAG SEGMENTATION</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Norvig’s Algorithm</td>
<td>20</td>
</tr>
<tr>
<td>3.2 An Extension of the Existing Methodology</td>
<td>24</td>
</tr>
<tr>
<td>3.3 Operational Issues</td>
<td>25</td>
</tr>
<tr>
<td>3.3.1 Web-Based Results: A Dead End</td>
<td>25</td>
</tr>
<tr>
<td>3.3.2 Twitter-Based Results</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Technical Issues</td>
<td>28</td>
</tr>
<tr>
<td>3.5 Methodology</td>
<td>29</td>
</tr>
<tr>
<td>3.5.1 Hashtag Acquisition</td>
<td>29</td>
</tr>
<tr>
<td>3.5.2 Text Acquisition, Preparation, and Training</td>
<td>31</td>
</tr>
</tbody>
</table>
3.5.3 Data Treatment

3.5.3.1 Database Setup

3.5.3.2 Segmentation

3.5.4 Evaluation

4 RESULTS AND DISCUSSION

4.1 Presentation of the Findings

4.2 Discussion

4.2.1 Error Analysis

4.2.2 Improvements for Future Research

4.2.3 An Application of This Work

REFERENCES

APPENDIX

A SEGBASE MODULE

B GET_HASHTAGS MODULE

C GET_TEXT_DATA MODULE

D INIT_DATABASE MODULE

E SEGEXT MODULE

F UTILITIES MODULE
LIST OF TABLES

Table 2.1. Sample Hashtags Segmented by the Baseline Algorithm..........................................17
Table 3.1. Relation Schema .......................................................................................................35
Table 4.1. Contingency Table for McNemar’s Test on Evaluation Results (No Discards) ..........................................................................................................................41
Table 4.2. Contingency Table for McNemar’s Test on Evaluation Results (with Discards) ..........................................................................................................................42
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>A sample tweet</td>
<td>2</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>Sample tweet revisited</td>
<td>14</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Sample trending topics for (from left to right) “World”, “United States”, and “San Diego” Twitter locales, respectively</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Tailored trending topics list example</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Sample trending topics for “New Orleans”</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Overview of workflow</td>
<td>30</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Performance of segmenters on hashtags</td>
<td>40</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

My heartfelt thanks must be extended to Rob Malouf, Mark Gawron, and André Skupin, all of whom provided much-needed guidance in the course of writing this paper, with special gratitude given to Rob, who gracefully put up with more formless nonsense than he had any obligation to. I would also be remiss for not thanking Peter Norvig for his very accessible writing and code that inspired this project in the first place. Finally, this would not have been possible without the support of my wife, Wendy.
CHAPTER 1

INTRODUCTION

Millions of Twitter users take advantage of Twitter’s ability to broadcast information pertaining to whatever topics they choose; likewise, millions of Twitter users receive an unending flood of undercategorized information. Computer-mediated social networking services such as Twitter are increasingly popular and, due to the vast amount of textual data contained in them, are also increasingly relevant to linguists and other members of the Natural Language Processing (NLP) community. The specific issues that such social networks present are diverse and challenging; this work is an attempt to name, quantify, and resolve one such challenge, i.e., the task of implementing a means of extracting more data from Twitter communications.

1.1 AN OVERVIEW OF TWITTER MECHANICS

Twitter is a popular computer-mediated social networking service, with 100 million active users as of September 2011 (Twitter 2011). Such services may be partially defined as microblogging platforms, in which users issue relatively short posts (sometimes referred to as STATUS UPDATES) pertinent to their thoughts, activities, and interests. In Twitter, the user Alice may FOLLOW the user Bob in order to receive Bob’s status updates in her STREAM, which consists of status updates from all users that Alice follows, listed in reverse chronological order. Users are connected asymmetrically, meaning that Bob is not obligated to follow Alice in order for her to follow him. Bob counts Alice among his FOLLOWERS, which are the set of users that receive Bob’s status updates.2 Although Bob may configure his

---


2 It is assumed here and for the rest of this paper that a user is synonymous with a Twitter account and not necessarily with a human being. In fact it is quite possible for a human to maintain more than one account, and accounts may be solely automated bots. However, in cases in which I attribute any kind of sentient action to a (human) user I trust that my meaning will be clear.
Twitter privacy settings such that only his followers are able to access his status updates, the default and common practice is for users to allow open access to their status updates, including the $n$ most recent status updates for some $n$ determined by Twitter$^3$. Users may actively inspect another user’s open profile and status update history in order to determine if they would prefer to begin passively receiving that user’s status updates.

Any particular status update is referred to as a TWEET. (Please refer to Figure 1.1 for an example.) Twitter’s prominence in popular culture is sufficient such that the term tweet has been verbified; therefore, a user may tweet a tweet. Tweets are primarily marked by their orthographic brevity: by design, tweets may be no more than 140 characters long. This constraint was originally a reflection of Twitter’s early conceptualization as a means of communicating over SMS (Sagolla 2009)$^4$ (Short Message Service, a common means of exchanging text-based messages between cell phones) but has remained a feature despite Twitter’s transition to a primarily Web-based platform. (As of this writing, Twitter does still offer SMS as a medium in which to receive and post tweets; however, this usage is largely deprecated due to its tendency to overwhelm users with constant incoming stream updates.)

![Figure 1.1. A sample tweet.](image)

Twitter offers a number of discourse mechanisms to their users which help enrich the Twitter user experience. Chief among them are the use of direct messages between two users; replies and mentions, both of which utilize the @ symbol; and the use of hashtags, which utilize the # symbol. Additionally, tweets can be disseminated through a process known as retweeting.

---

$^3$ Throughout this paper, I generically refer to the social networking service, the APIs for interfacing with it, and the company maintaining it as “Twitter”.

$^4$ http://www.140characters.com/2009/01/30/how-twitter-was-born/ Published January 30th, 2009 and retrieved May 16th, 2012.
DIRECT MESSAGES are tweets that are intended to bypass the inherently open architecture of Twitter’s social network. Assuming the prior asymmetric relationship between Bob and Alice, Bob may send a direct message to Alice which is not viewable by anyone except them. Alice may not send such a message to Bob unless their relationship gains symmetry.

MENTS are a feature of tweets which contain a username; if user Bob with username bob is mentioned in a tweet, the @ symbol is typically prefixed to his username to produce @bob. Formatted thusly, any instance of @bob in any tweet is tracked by Twitter, which is then able to show Bob all public tweets which mention him. REPLIES are a special case of mentions that are predicated on the form @username being the first term in a tweet, and are intended to facilitate an open conversation. For example, if @carol and @bob both follow each other and @carol issues a tweet, @bob may begin a responsive tweet with “@carol” in order to remove that tweet from his follower’s streams. Exceptions to this removal behavior are that Carol will see that responsive tweet in her stream despite being Bob’s follower; additionally, if @alice follows both @bob and @carol, Bob’s “@carol”-initial tweet will appear in Alice’s stream as well. If a follower navigates to Bob’s profile, however, she will see the history of all of Bob’s tweets, including those that did not appear in her stream. Equivalently, Twitter provides an API method to view Bob’s (or any user’s) tweet history (Twitter 2013).² Despite their name, replies are often used to initiate an open conversation; in such a case, the previous example applies with the exception of @carol’s initial tweet not being issued: @bob sends a tweet beginning with “@carol” and @carol responds by beginning a tweet with “@bob” for some number of iterations, and since @alice follows both she is privy to the conversation.

Topics or keywords in tweets can optionally be explicitly marked through the use of HASHTAGS. Hashtags are strings prefixed by the # symbol, in the form #hashtag. The hashtag may be composed of one or more terms; if the latter, the terms are concatenated to form a continuous string. Users utilize hashtags to categorize their own tweets and to search for

---

⁵ More precisely, “@” must be the first character of the tweet, followed by the username.

⁶ https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline Published March 7th, 2013 and retrieved November 4th, 2013.
tweets containing a particular hashtag. Although hashtags may occur at any point in a tweet, user behavior shows a pronounced tendency to place them at the end, helping to differentiate them from the syntax of the remaining tweet text. The use of hashtags was originally proposed by Chris Messina (2007) who adapted the idea from Jaiku, an earlier social network that used hashtags to organize groups according to interests. Their broader use emerged organically from Twitter’s user base (Twitter n.d.b), and they continue to evolve (Orlean 2010): in addition to topic, they may represent commentary and/or sentiment with respect to its containing tweet and may even reflect a user’s awareness of the discourse structure of Twitter as a whole.

Finally, RETWEETS are a special case of tweets which take advantage of Twitter’s network structure to gain an audience larger than was originally accorded them. By way of example, suppose that @carol follows @bob but not @alice, and that @bob follows @alice. A tweet that @alice issues will initially be received by @bob but not @carol. @bob may choose to retransmit @alice’s tweet as his own with minimal effort and thereby cause his followers, including @carol, to receive @alice’s tweet as his retweet. (Retweet has been verbified in the same fashion as tweet.) Optionally, @bob may give explicit credit to @alice for the original tweet by prefixing the retweet with the marker “RT” followed by a space and the amended username “@alice:” (note the suffixed colon). However, this is not always done, especially in cases in which the text of the original tweet is near or at 140 characters. Implicit credit to the original author of the tweet is always provided by the Twitter platform in some fashion.

There exists as of this writing a small number of other Twitter-specific discourse markers, but they are not relevant to this work and will not be discussed.

---


1.2 General Motivation for This Work

Although Twitter’s long-term viability as a member of the SNS phenomenon probably cannot accurately be extrapolated from current popular usage, the modes of communication that it has helped engender seem likely to have staying power. Society continues to evolve toward a norm of constant individual availability. This availability extends not only to the burden and responsibility of accepting input – voice messages, emails, SNS communications, text messages, etc. – but also to the expectation and/or opportunity of output: the ability afforded by such technologies to succinctly communicate key ideas and relay compacted information in short bursts on a continuous and ad hoc basis.

The specifics of how information is compacted in tweets begin to get at the motivation for this research. The inherent terseness of a given tweet constrains the number of topics within; analysis and commentary thereof is likewise abbreviated. Yet one may wish to reference a related topic or express some kind of sentiment regarding the tweet topic(s). The nature of such parsing is roughly approximated by machine learning classifiers: the classifier observes the presence or absence of certain features (i.e., attributes) in a data set, and then compares the distribution of such features to those found in already-known data; it then categorizes that data set and, in moving that data set into the group of already-known data, learns from it. So too do humans observe their surroundings and other input and learn from such observations; the obvious difference being that humans are able to concurrently recognize and parse features that classifiers are not aware of and thereby derive a more accurate understanding of their observations than classifiers are.

The motivation for this research lies in the desire to extract more features from tweets in an attempt to lay the groundwork for further examination of whether the added features may help in classification techniques. As a brief example, consider the task of topic modeling. Techniques for assessing the underlying topic(s) of a document invariably rely in some way on counting features from a supplied corpus. For example, Latent Semantic Indexing (LSI) transforms a corpus into a k-dimensional space in which each dimension represents an underlying, or latent, topic that is described by terms and their weighted frequencies (see Deerwester et al. 1990). This technique allows one to quantify how related a set of test terms are in the context of a training corpus. Crucially, it also allows a document to be viewed as a soft mixture of topics; this means that comparison between documents need
not be based on a strict assignment of a single topic to a given document, but rather that a
given document can be said to be “more about” a topic than another document, despite one or
more other topics being shared by the two documents. Such fine-grained filtering of
documents depends on a sufficiently large $k$, which in the case of modeling tweets may be
effected by adding sufficiently many terms (Bradford 2008).

Furthermore, suppose that a corpus of tweet text (including hashtags) annotated for
positive and negative sentiment is available. A novel tweet containing a hashtag not observed
in training must be classified. The novel tweet could be classified on the basis of its text and
the new, unsegmented, hashtag; which is to say that the hashtag as a whole would be
considered a feature in this case. It would be a simple matter for a memory-based classifier to
compare the text of the novel tweet with that of the training corpus and decide on a sentiment
for the tweet, and by eventual extension, the hashtag. A variant of this procedure might see
the novel tweet text being augmented by text from tweets outside of the training set. These
tweets contain the same hashtag yet still are treated as a single unified document by the
classifier. This approach has the benefit of adding features (in this case tokens) to the test
tweet, which would putatively assist the classifier in categorizing it. The intuition in this case
is that a hashtag associated with a given topic would be correlated to words associated with
same; more instances of this relationship seen in the text makes for a more definitive
classification.

Having mentioned the benefit of adding features to the classifier, the question arises
of whether more features can be eked out of a given tweet. One possibility of doing so
appears trivial: decompose the tweet’s hashtag(s), if existing, into its contained words; the
individual words are then treated as features along with the words in the tweet text proper.
One objection that could be made to this method is that there simply is no need; if the tweet
text can be augmented as described previously, it is unlikely that a segmented hashtag would
add informative features beyond those provided by the additional text. But suppose that the
test tweet contains the only known instance of a particular hashtag. In such a case, one might
augment by stripping the tweet of stopwords and treating the remaining text as a query or
queries to the general Twitter collection. The result set could then be tokenized and treated in
the usual manner. Alternatively, suppose that a hashtag contains a topical reference that is
salient to current cultural discourse. As we shall see, current segmentation techniques may
fail to detect neologisms. In this case, even if it is possible to successfully classify the tweet’s topic and/or sentiment based on the text alone, information has been left on the table, so to speak, by the failure to associate a topic and/or sentiment with the neologism within the hashtag. One desires to continually update the model being used.

1.3 **RELATED WORK**

This research utilizes work done in a range of fields. Whereas informatics and other elements of Web research have an established history, due to the relatively brief amount of time that computer-mediated social networks of the sort represented by Twitter, Facebook, etc., have existed, academic interest in the text processing issues that they pose has only recently begun to develop. What follows is a brief overview of these topics; many issues that are touched upon will be revisited in more depth.

1.3.1 **The Web as Corpus**

As will be shown in Section 3.3.1, one possible area of generating training data for hashtag segmentation involves utilizing the web as a general resource. At least as early as 1998, the use of the web as a corpus was advocated by Grefenstette (1998), in his case for the purposes of machine translation. Resnik (1999) and others soon voiced support for this approach, which proved to be fruitful in a variety of applications: anaphora resolution (Modjeska et al. 2003), word sense disambiguation (Santamaría et al. 2003), and language modeling (Zhu & Rosenfeld 2001), among others. This last task is of particular note. Nakov and Hearst (2005) cited Keller and Lapata’s (2003) work in commenting that “the most popular use of the Web as a corpus is as a means to obtain page hit counts as an estimate for n-gram word frequencies.” However, that approach turns out to be suboptimal. In Keller and Lapata 2003, the authors describe their process for obtaining bigram frequencies as a simple matter of counting results returned from a query to a search engine. Nakov and Hearst (2005) summarized several arguments against this method: the related issues of dynamic and “dancing” results, and the fact that the returned number of search results are oftentimes rounded.
1.3.2 Twitter as a Medium of Communication

Java and colleagues (2007) were among the first to rigorously examine some features of Twitter. By utilizing network-theoretic methods to study certain topological properties of Twitter’s social network, they derived a typology of Twitter users that featured three main categories of information sources, information seekers, and generalized friends. Additionally, they argued for a classification of user intent over individual tweets, and described four main categories of such intent: daily chatter composed of one-off tweets, conversations arising from such chatter and which utilize the @ symbol to name participants, and information sharing by disseminating URLs and/or updates on current events. Naaman and colleagues (2010) extended this user and tweet analysis and found a bifurcated user behavioral structure based on tweet content, concluding that while some users are primarily information sources, a majority of users post tweets of a more self-centered nature. More recently, Bandari and colleagues (2012) examined a different aspect of social media: that of its role in propagating online news items. They collected a data set of articles via a news feed and then found the number of times each article was linked to on Twitter. From a training subset of the articles, they also identified a number of features that considered together were found to be the most relevant in terms of predicting an article’s Twitter popularity, as measured by the number of times its link was included in a tweet. In testing, they achieved an overall accuracy of 84% success in classifying an article’s membership in one of three popularity tiers. They further found that one of the classification features, the category labels assigned by the news feed agent to any article, was overall suboptimal in the prediction stage. This was largely caused by the interrelated nature of the categories, which led to articles that could be argued to be miscategorized. They also discovered that particularly in the case of predicting zero-tweet articles, the features of named entities within an article and its subjectivity did not provide very much, if any, prediction information.

1.3.3 Existing Segmentation Techniques

In Segaran and Hammerbacher (2009), Peter Norvig (2009) proposes an efficient method of word segmentation. Norvig begins by pointing out that although English orthography typically marks word boundaries, there are some edge cases, such as URLs, in
which this does not hold; how, he asks, could such cases be programmatically accounted for? He outlines a basic procedure as follows:

1. Define a probabilistic language model, i.e., a probability distribution over all possible English words. This model is trained on the unigrams subset of the trillion-word corpus published by Brants and Franz (2006) of Google and has a method for handling words not found in the training set.

2. Establish a candidate set of all possible segmentations for a given string in the test set. This is essentially a modified powerset of the string characters, constrained by character contiguousness.

3. For each member of the candidate set, compute the independent probabilities of each segment in the member by comparing it to the language model on an as-needed basis and then multiply these probabilities together to calculate a segmentation probability. Select the member with the highest segmentation probability as the correct segmentation.

Although Norvig does not specifically mention segmenting hashtags as an application of his algorithm, it is obvious and trivial to do so. A somewhat less trivial extension of his algorithm involves the creation of a more accurate model. Norvig does explore modifying the basic algorithm to include members of the bigrams subset of the trillion-word source corpus and finds it to be slightly more accurate, but one intuition regarding the problem of segmenting hashtags is that they may not be helped by such modifications due to their general brevity. Another more pressing realization is that the static nature of the source corpus, perhaps more than anything else, is a potential roadblock to a more accurate hashtag segmentation model. Given that hashtags so often reference topical events, a key supposition of this research is that some form of online corpus updating is needed to increase the accuracy of a hashtag segmenter.

Although this research was initially inspired by Norvig’s work, the field of segmentation of hashtags, web domains, and other strings is currently active, and indeed Norvig himself drew on existing work. The general task of segmenting words has a rich history, particularly in the domains of various unsegmented languages, such as Chinese. Palmer (2000) provides a comprehensive overview of the various challenges in text

---

10 As detailed in the brief literature review that follows, in fact the algorithm Norvig presents is heavily indebted to pre-existing techniques; however, it will herein simply be labeled as his due to his presentation of it being a starting point in my research as well as for ease of reference.
segmentation and tokenization (chief among them the well-known issues inherent in specifying what constitutes a word in a given context) and notes that a popular approach to word segmentation utilizes a variant of GREEDY SEARCH known as “maximum matching”, or MAXMATCH. He describes MAXMATCH as traversing a desegmented string and choosing the longest possible substrings that are words, relying on a known vocabulary. In the case of the table down there being desegmented to thetabledownthere, MAXMATCH would find initial segmentation candidates the and theta; due to its greediness, the ultimate segmentation would be theta bled down there. As we will see, a probabilistic model typically avoids such segmentations. Specifically, Sproat and colleagues (1996) reviewed previous literature and concluded that prior efforts in Chinese word segmentation could be generally categorized as purely statistical, purely lexical rule-based, or a hybrid of the two. They went on to propose such a hybrid that directly informed Norvig’s approach.

Mejova and Srinivasan (2012) found that joint probability modeling from multiple corpora spanning distinct domains performed significantly better than modeling on any one corpus, an insight shared in this work via its use of corpora personalized from many sources for a given hashtag. However, they assigned weights to each corpus used via supervised learning, which, while producing impressive results, is not conducive to anything approaching the real-time segmentation pursued by this work. Li and colleagues (2012), on the other hand, acknowledge the importance of tweet time-sensitivity, although in the context of tweet text segmentation in pursuit of a more powerful named entity recognition (NER) system than is currently used.
CHAPTER 2

THE PARAMETERS OF TWITTER

Twitter belongs to the class of computer-mediated social networking services that have evolved from the BBS ecosystem (Goble 2009)\footnote{http://www.digitaltrends.com/features/the-history-of-social-networking/ Published January 21st, 2009 and retrieved May 23rd, 2012.} of the internet’s early days to the current environment dominated by services predicated on a network architecture. As Naaman and colleagues (2010) noted, these services, exemplified by Twitter, Facebook, and others, constitute a new type of communication technology that is distinguishable by three factors: (a) the semi-public characteristics of the discourse, (b) the brevity of the discourse content, and (c) the network-driven nature of the discourse. Hereinafter these services will be generically referred to as Social Network Services (SNSs), after boyd and Ellison 2007.\footnote{Although the authors of the cited work defined SNSs as Social Network Sites (rather than Social Network Services), I prefer the latter for its inclusiveness of the varied forms they may take, such as mobile applications built on a website’s APIs.}

Although the reader is encouraged to refer to previous academic research pertinent to the unique aspects of these emerging communication systems, a brief examination of some specific aspects of Twitter is given; such an overview will highlight some issues inherent with the research that is being discussed in this work.

2.1 A USER TYPOLOGY

In this work I adapt Java and colleagues’ (2007) broad user classification, which they arrived at via topological analysis of the Twitter network, by supplementing it with observed sub-types. Since, as that group observed, individual users are known to display multiple roles and intentions in this still-evolving SNS, a strict and thorough classification of users and their intentions is not possible. However, a cursory overview of such will give the reader a frame of reference for the data gathered in this work. Future research in hashtag segmentation or
general text processing of tweets may build off of user type classifications and other such parameters of Twitter.

The largest sub-section of Twitter users a prototypical Twitter user is connected to (by either following them or being followed by them) can be classified as “friends”, which is to say, people that she knows outside of Twitter. As with any SNS, this collection of users may potentially break down into separate and perhaps overlapping spheres: family, co-workers, good friends, colleagues, acquaintances, etc.

Another notable category of user what Java and colleagues called an “information source”, defined as a user who has a large number of followers that find that user’s posts valuable due to their informative and/or frequent nature. There are several variants of this type, all of which can be loosely classified as being “corporate” or “non-corporate”. In the former, it is common to see news organizations, bloggers with organizational affiliations, and impersonal representatives of an organization. Bots which automatically perform tasks such as retweeting and link-sharing may be included in this group as well. To the non-corporate variant belong independent bloggers and other content producers, celebrities, and so on.

One other type of user is the “information seeker”: a user who rarely tweets but does view her stream as informative and worthy of attention. By definition, this type of user will be under-represented in the data collected for this work.

Java and colleagues also classified a number of user intentions. The most notable among them included open conversations, chatter, sharing information via links, and reporting and/or commentary on current events. They found that chatter – routine updates of quotidian details – comprised the majority of tweet content, and that roughly 12% of tweets extended into a conversation.

Users are not constrained to one typological category. For example, if Alice identifies Bob as an information source only, that does not preclude Carol from identifying Bob as a friend, and possibly also as an information source. Likewise, users may freely assume different intentions between tweets or even in a single tweet. Bob may, for instance, share a link pointing to a media account of a current event and offer commentary on the current event, the story, or both, all in one tweet, and then participate in a conversation with Alice that follows up on the tweet content.
2.2 Tweet Structure

Tweets are exceedingly brief, even by the standards of the microblogging communication platform offered by today’s SNSs. O’Connor and colleagues (2010) found that the average tweet is 11 words in length. This brevity belies the depth of content that a tweet may exhibit and the variety in which it is exposed. In the sections that immediately follow I briefly discuss some compositional issues and features of tweets in order to help define the tweet corpus used in this research.

2.2.1 A Sample Tweet

Figure 2.1 is a tweet representation that highlights several aspects of tweets. The user account is @slate, the official Twitter account of Slate magazine, a popular news and entertainment destination on the web. Its Twitter profile reads in part “Politics, culture, technology, business, news, and commentary. Procrastinate better.” The user type can therefore be identified as a corporate news organization acting as an information source. It follows that most or all of the tweets from this account fall under the rubric of reporting and commentary on current events, often via links to news stories on the Slate website or elsewhere, and that is what is seen in this example. Note also the stylistics of the commentary: by utilizing a special character at the beginning of the text, the tweet is able to reference @BusinessInsider in a tweet-initial position without being formatted to only be visible to @BusinessInsider and the group of users that happen to follow both @slate and @BusinessInsider. Finally, and intriguingly, the tweet utilizes a memetic hashtag which, as of this writing, is associated with a particular style of domination through personality, usually via the media. Through the utilization of this hashtag one may infer certain aspects of the tweet content and of @BusinessInsider that might otherwise not be apparent and would likely in any case require more than 140 characters to flesh out.

13 As of December 18th, 2012, this account follows 348 other accounts, is followed by 482,363 accounts, and has issued 36,173 tweets.
Figure 2.1. Sample tweet revisited.

### 2.2.2 Hashtags

Twitter uses proprietary algorithms to surface trending topics, which may not be represented by hashtags. Despite not divulging their methodology for identifying trending topics\(^{14}\), Twitter has stated that key metrics for the algorithms include novelty as well as popularity (Twitter 2010).\(^{15}\) This means that the longer a topic has been trending, the higher a bar it must clear with respect to popularity. Additionally, there is a spatial component to trending topics that is hierarchical in nature: Twitter’s awareness of many of its users’ approximate geographical locations allows it to categorize trending topics (if available) on a granular level. For example, topics that are trending from users within a city contained by a given country may be almost or completely disjoint from those trending from users countrywide.

Figure 2.2 shows a variety of trending topics from different locations, captured within minutes of each other on December 20\(^{th}\), 2012. One notices that the worldwide trends list is multilingual and otherwise differs slightly from the other lists; the United States and San Diego trends lists, on the other hand, happen to be identical. In this example, each of the three lists coincidentally contains five hashtags out of the nine entries within, and on each list the hashtags are (again coincidentally) listed before the other topics.

Compare this to this author’s personalized trends list as shown in Figure 2.3, automatically curated by Twitter.

---

\(^{14}\) On November 1\(^{st}\), 2012, researchers at MIT announced that they had successfully reverse-engineered Twitter’s trending algorithm. See [http://web.mit.edu/press/2012/predicting-twitter-trending-topics.html](http://web.mit.edu/press/2012/predicting-twitter-trending-topics.html)

\(^{15}\) [http://blog.twitter.com/2010/12/to-trend-or-not-to-trend.html](http://blog.twitter.com/2010/12/to-trend-or-not-to-trend.html) Published December 8th, 2010 and retrieved December 20\(^{th}\), 2012.
Figure 2.2. Sample trending topics for (from left to right) “World”, “United States”, and “San Diego” Twitter locales, respectively.

Figure 2.3. Tailored trending topics list example.

This personalized trends list, captured in the same time frame as those in Figure 2.2, shows that the ordering of hashtags within a trending topic list is not a given. Additionally, the ratio of hashtags to overall trends is not constant. Both stipulations are seen in Figure 2.4:

To help gauge the relationship of hashtags to trends, over the span of a few hours I ran a script that collected 488 trends from 49 locations derived from the geographic and/or political boundaries of the United States; after duplicates had been discarded, 42 hashtags were identified out of 91 trends, meaning that approximately 46% of the available trends were found to be in the form of hashtags. The extent to which this percentage is a function of Twitter’s trending algorithm is currently unknown.

On the afternoon of December 21st, 2012, I perused the trending topics lists that Twitter offers of some (primarily English-speaking) locations and selected a small number of hashtags. I made no attempt to apportion the hashtags such that the set would display the
Figure 2.4. Sample trending topics for “New Orleans”.

proportions of correct segmentations that I expected to see in testing. Rather, I expected the Norvig segmenter to successfully handle about half of the set, thereby giving a sketch of some features of its language model. In Table 2.1, I present a number of segmentation test cases (the majority of which are drawn from this hashtag set), and detail my naïve expectations of segmentation success\textsuperscript{16}, as well as the actual success, the segmented hashtag, and any notes regarding that hashtag.

From Table 2.1 we see that at least anecdotally, there is cause to believe that one can segment hashtags better than the Norvig segmenter does. Terms that did not exist before the source data collection, such as Instagram and hashtag, are not handled well; neither is a term such as X Factor, which was segmented as XFactor. (In fairness, xfactor occurs in the source data 25,791 times and its presence may be an artifact of data cleaning procedures.) Note also that the term vicmas, a BLEND that combines a shortening of the name Victoria and the term Christmas, is not recognized despite what would seem to be a fairly easy segmentation between it and day 1 in VicmasDay1.

\textsuperscript{16} The justifications of these expectations are provided in the Notes section of Table 1. Giving my expectation of success in the context of the baseline segmenter’s performance is a rough heuristic for the accuracy of that segmenter, and begins to indicate situations in which an improved segmenter is needed. More precise figures are given in Section 4.1.
<table>
<thead>
<tr>
<th>Hashtag</th>
<th>Actual Success</th>
<th>Expected Success</th>
<th>Segmentation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#choosespain</td>
<td>Y</td>
<td>Y</td>
<td>choose span</td>
<td>This was not actually a hashtag; Norvig presented ‘choosespain’ as a test case in Segaran and Hammerbacher (2009) and I include it here to verify the implementation of his algorithm.</td>
</tr>
<tr>
<td>#e3followspree</td>
<td>Y</td>
<td>Y</td>
<td>e3 follow spree</td>
<td>Although grepping “e3” in Norvig’s unigrams file reveals that it is not in the model, it remains a viable segment due to the frequency of the other two segments.</td>
</tr>
<tr>
<td>#QBsBetterThanSanchez</td>
<td>Y</td>
<td>Y</td>
<td>qbs better than sanchez</td>
<td>Although one might not expect either ‘qbs’ or ‘sanchez’ to be in the model, they occur with frequencies of 144,237 and 2,847,729, respectively. Regardless, due to the frequency of ‘better’ and ‘than’, segmenter success was assumed.</td>
</tr>
<tr>
<td>#NRA</td>
<td>Y</td>
<td>Y</td>
<td>nra</td>
<td>Verification that many common acronyms and initialisms occur with sufficient frequency to ensure accurate segmentation.</td>
</tr>
<tr>
<td>#7FactsAboutMyBestfriend</td>
<td>Y</td>
<td>Y</td>
<td>7 facts about my best friend</td>
<td>Hashtags occasionally utilize camel case to aid in human parsing, as seen here. This example is notable for …?Bestfriend, which the segmenter handled as the more acceptable ‘best friend’.</td>
</tr>
<tr>
<td>#TimelessChristmas</td>
<td>Y</td>
<td>Y</td>
<td>timeless christmas</td>
<td>The joint probability of ‘time’ and ‘less’ is less than the probability of ‘timeless’, according to the model.</td>
</tr>
<tr>
<td>#reasonsobamaisslate</td>
<td>Y</td>
<td>Y</td>
<td>reasons obama is late</td>
<td>Although ‘obama’ is current and topical as of this writing, it was in the general lexicon at the time of the corpus collection, as well.</td>
</tr>
<tr>
<td>#newtown</td>
<td>Y</td>
<td>N</td>
<td>newtown</td>
<td>Due to lack of knowledge in the domain of place names, I expected the segmenter to treat ‘newtown’ as ‘new town’; however, ‘newtown’ is seen 1,191,486 times in the unigrams file, which is sufficient to return the correct segmentation.</td>
</tr>
</tbody>
</table>

(table continues)
<table>
<thead>
<tr>
<th>Hashtag</th>
<th>Actual Success</th>
<th>Expected Success?</th>
<th>Segmentation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#gangnamstyle</td>
<td>Y</td>
<td>N</td>
<td>gangnam style</td>
<td>“Gangnam style” was an extremely popular meme in 2012, but my expectation that it would be virtually nonexistent in the model proved to be incorrect.</td>
</tr>
<tr>
<td>#instagram</td>
<td>N</td>
<td>N</td>
<td>insta gram</td>
<td>The photo-sharing SNS Instagram was formed in October 2010, well after the corpus data collection.</td>
</tr>
<tr>
<td>#VicmasDay1</td>
<td>N</td>
<td>N</td>
<td>vicmasday1</td>
<td>This was generated by neo-celebrity Victoria Justice (acting as a non-corporate information source), as part of a series of tweets enticing her followers to enter a contest via tweeting with this hashtag.</td>
</tr>
<tr>
<td>#jaydento44k</td>
<td>N</td>
<td>N</td>
<td>jaydento44k</td>
<td>This was generated by neo-celebrity Jayden Sierra (acting as a non-corporate information source), asking his followers to retweet or otherwise use this hashtag as a means of enhancing brand awareness, with the goal of reaching at least 44,000 followers.</td>
</tr>
<tr>
<td>#SDSUvsBYU</td>
<td>N</td>
<td>Y</td>
<td>sd suvs byu</td>
<td>Not all initialisms occur with sufficient frequency so as to guarantee correct segmentation.</td>
</tr>
<tr>
<td>#XFactorFinale</td>
<td>N</td>
<td>Y</td>
<td>XFactor Finale</td>
<td>At the time of this writing, “The X Factor” is a popular international televised singing competition. My expectation was that ‘X’ and ‘Factor’ would individually be adequately represented in the model to surpass the probability of ‘xfactor’.</td>
</tr>
<tr>
<td>#ReplaceMovieTitleWithSarooof</td>
<td>N</td>
<td>Y</td>
<td>replace movie title with 5aro of</td>
<td>This hashtag utilizes the Arabic chat alphabet (“Arabish”, or “Arabizi”), an organically generated character encoding from Arabic to Latin script that allows the use of Latin numerals in transliteration. While I did not expect the language model to handle ‘Sarooof’, I did expect it to be segmented as ‘5 a roof’.</td>
</tr>
<tr>
<td>#HashtagAskRyan</td>
<td>N</td>
<td>Y</td>
<td>hash tag ask ryan</td>
<td>In an ironic turn, the segmenter viewed 'hashtag' as 'hash tag'.</td>
</tr>
</tbody>
</table>
CHAPTER 3

A METHOD OF IMPROVED HASHTAG SEGMENTATION

I have argued that although a non-segmented hashtag can be a feature, there are cases in which it may benefit a classifier to segment a hashtag into individual features. However, if there is an unrecognized term in a hashtag that would cause a segmenter with a static source corpus to improperly segment it, we may be better off not segmenting at all. How do we increase our confidence that a hashtag is properly segmented?

One method is updating the source corpus of the segmenter, and therefore our language model, so that it reflects the most current English language usage possible: the slang, the surprising new topics, the heretofore obscure but newly relevant names of people and places, and so on. Then if a candidate term can be shown to be sufficiently statistically significant, it can be accepted as a word. But how, exactly, can the model be reliably updated? The original Brants and Franz corpus released in 2006, known officially as the “Web 1T 5-gram Version 1” corpus but referred to informally as the “trillion-word” corpus, contains 1,024,908,267,229 tokens; its frequency distribution of unigrams holds 13,588,391 types, or entries. It totals 24 GB of compressed data. Although there does not appear to be much information regarding the trillion-word corpus’ parameters of data collection publicly available, in 2009 Brants and Franz released the “Web 1T 5-gram, 10 European Languages Version 1” corpus. That corpus, which is approximately 28 GB compressed, is reported by them to have been gathered between October and December of 2008. It hardly needs be said that this scale of data collection is not easily replicable with current consumer-grade technology, and certainly not within the timeframe required by the task before us. If a motivation for segmenting hashtags is to classify sentiment regarding topical events, a data collection timeframe of months will not suffice.

In the pages that follow, I lay out a segmentation procedure that attempts to gracefully handle the time and topicality constraints necessitated by hashtag input while
maintaining the speed and accuracy of Norvig’s existing algorithm. I begin with a description of the existing procedure.

3.1 NORVIG’S ALGORITHM

In Chapter 1 I touched upon the segmentation method outlined by Norvig in Segaran and Hammerbacher (2009). Following him, I summarized his algorithm as first training a probabilistic language model, then enumerating a candidate set of segmentations for a test string, and finally selecting the most probable candidate based on probabilities derived from training. We will now explore these steps in further detail, as this research is a further application of his algorithm.

In this research, I work with the unigrams corpus also used by Norvig. Rather than directly using the approximately 13-million-type unigrams corpus included in the trillion-word corpus, Norvig created a cleaner version by removing all tokens that did not consist solely of alphabetic characters, combining entries by creating case-insensitive types, and reducing the number of entries to the top 333,333 most common types. He found that these entries sufficed to retain 98% coverage (i.e., the summed count of the final 1/3 of a million types, or words, represented 98% of the total token count given in the earlier version) while reducing the corpus size to 5 MB. Note that this does present a challenge for hashtag segmentation from the outset: as a given hashtags may easily include digits, we are relying on these digits to be a natural single group (such as a date) that the base segmenter will separate from the other text by default, or else rely on the extended segmenter to recognize and correctly handle more complicated cases.

When training a language model for segmenting, we must consider how to map real-world factors to something quantifiable. A model, by definition, abstracts away unimportant aspects of the system that it simulates. In the case of our language model, we gloss over all the context inherent in a productive language system with a frequency distribution; i.e., the language’s morphology, syntax, orthography, semantics and other linguistic factors are reduced to an accounting of its terms. Following Norvig’s example, given the string choosespain (derived from a URL), we as humans apply our world knowledge to hopefully parse it as choose spain rather than chooses pain. This knowledge is deep, broad, and often unspoken, but we rely on it to choose the former as being a more probable interpretation. A
language model based on our source corpus has no proximate access to that knowledge, but it
does ultimately have access to the quantifiable consequences of that knowledge, namely that
the term *choose spain* is more probable than *chooses pain*.

The model’s terms need not include only bigrams such as *choose spain*; the trillion-
word corpus holds unigrams through five-grams. Using this *N*-GRAM data, we can treat our
language model as a MARKOV model, and estimate the probability of a word sequence *W*, or
any word *w* within it, given a history *h*. Generically, assuming a sequence of random
variables *X*₁ ... *X*ₙ, the probability of such a sequence can be determined by the chain rule:

\[
P X₁ ... Xₙ = P X₁ P X₂ X₁ P X₃ X₂ ... P Xₙ Xₙ₋₁
\]

\[
= \prod_{k=1}^{n} P(X_k|X_{k-1}^{k-1})
\]

Given this, for the word sequence *W* of length *N* being represented as *w*₁ⁿ, we see:

\[
P w₁ⁿ = P w₁ P w₂ w₁ P w₃ w₂ w₁ ... P wₙ wₙ₋₁
\]

\[
= \prod_{k=1}^{n} P(w_k|w₁^{k-1})
\]

However, if *N* is sufficiently large, the productivity of language dictates that for some *n*,
*P(w₁ⁿ)* is zero, which in turn causes our joint probability estimate to become zero as well.
Hence we apply the Markov assumption, which states that the probability of a word *w* can be
approximated by setting *N* to be relatively small:

\[
P w_n w₁ⁿ⁻¹ \approx P w_n w₁ⁿ⁻₁
\]

For example, given the sequence Top Congressional leaders met with President
Obama, we can calculate the probability of Obama as *P Obama President* rather than
*P Obama Top Congressional leaders met with President* by using a bigram model,
which estimates the probability of a word given all previous words *P wᵱ w₁ⁿ⁻¹* by
computing the conditional probability of the single most recent word *P wᵱ wₙ₋₁*. It follows
that the probability of the whole sequence according to a bigram model is:

\[
P(w₁ⁿ) \approx \prod_{k=1}^{n} P(w_k|w_{k-1})
\]

or, allowing for special sentence-delineating characters that ensure that bigram contexts
across all words create a probability distribution (see Chen & Goodman 1996):
Let us consider the application of a bigram model to the task of hashtag segmentation. On the afternoon of December 29th, 2012, I examined my Twitter-curated personalized trending topics list and selected #MentionPerfection as a potential exemplar of how a hashtag might be described by a bigram model. Passing over the issue of how to insert <s> into an existing frequency distribution, we shall attempt to compute the bigram probability of perfection given an immediately preceding mention: \( P(\text{perfection}|\text{mention}) \). This is calculated by normalizing the count of the bigram mention perfection with the unigram count of mention:

\[
P(\text{w}_n | \text{w}_{n-1}) = \frac{C(\text{w}_{n-1}\text{w}_n)}{C(\text{w}_{n-1})}
\]

(This normalization is permissible due to the fact that the unigram count of mention is equal to the sum of all bigrams beginning with mention.) Manually calculating this maximum likelihood estimation then becomes a simple matter of grepping the bigram corpus for the count of mention perfection and the unigram corpus for the count of mention and dividing the former by the latter:

```
brandon@Mimisbrunnr:~/code/segmenter/corpora$ grep -nw "mention perfection" bigrams.txt
```

However, we immediately run into a problem: mention perfection does not occur in the bigram corpus. According to the bigram model, #MentionPerfection cannot exist!

To deal with this, we have three options. We could implement some sort of smoothing algorithm with our bigram counts by “stealing” some probability mass from existing bigrams to assign to bigrams that don’t occur in training. Alternatively and/or additionally, we could tweak our procedure to backoff to constituent unigram counts when bigram counts are not available and/or have a sufficiently low probability. However, as Norvig notes, in addition to the irritation of implementing these changes and adjusting the bigram corpus to allow <s> and </s>, there is also the consideration that \( N > 1 \)-gram data takes up more room in RAM than unigrams do. Additionally, as mentioned previously, a large number of hashtags are only a few words in length, if not only one word long. If we were to fully implement a backoff algorithm for hashtag segmentation that began with 5-
gram probabilities, as is allowed by the trillion-word corpus, it is unclear whether the additional computing costs would be worthwhile.

In his work, Norvig eventually settles on a bigram model, but he does not consider the effects of the minimal sequence length enforced by most hashtags; i.e., he does not test his segmenter on any of a class of concatenated strings that are constrained in length. Since many hashtags are in fact single terms and not in need of segmentation, a segmentation algorithm that relies on an assumption of more than one term within a string will fail. For that reason, I utilize his earlier method of simply relying on a unigram model, which neatly avoids some of the issues discussed above; the question of how to handle unseen words is given more attention later on in this work. Using the unigram model accommodates the existence of unigram hashtags, of which there are many, and hopefully generalizes to ngrams for some $n$. In this model, the probability of a word sequence $W$ of length $N$ is simply the product of the probability of each individual word:

$$P(w^n_1) \approx \prod_{k=1}^{n} P(w_k)$$

According to this model, the probability of #MentionPerfection would simply be the product of the probabilities of mention and perfection, both of which do occur in the unigram corpus:

```
brandon@Mímisbrunnr:~/code/segmenter/corpora$ grep -nw "mention" unigrams.txt
3935:mention 18291476
brandon@Mímisbrunnr:~/code/segmenter/corpora$ grep -nw "perfection" unigrams.txt
11450:perfection 4079350
```

What still must be considered, even in this simplified model, is a method of handling words not seen in the training corpus. (After all, there are no 0-grams to back off to.) For the sake of reproducibility, I again use Norvig’s method, which takes into account the number $N$ of tokens in the unigrams corpus and the relative probabilities of unseen words of various lengths in order to arbitrarily set the likelihood of an unknown word $<UNK>$:

$$P(<UNK>) = \frac{10}{N \times 10^{\text{length(<UNK>)}}}$$
However, the use of this method in my research is considered in the context of having earlier adjusted the training corpus in a manner that attempts to reduce the likelihood of finding unknown words; again, this will be revisited at a later point in this work.

Having trained on the unigrams corpus in order to generate our probabilistic language model, the next task is to segment a test string and choose a segmentation derived from the probabilities found in the model. Consider the possible segmentations of the word *cat*: they are *c|at*, *ca|t*, *cat*, and *c|a|t*. In this three-character string, there are two locations that may either or both hold a word boundary and it is also possible for the string to not be segmented; we thus see four segmentation candidates. This indicates a combinatorial guideline: for an *n*-character string there are \(2^{n-1}\) possible segmentations, since there is a binary word boundary modality for any of the \(n - 1\) positions between characters.

Norvig uses a *segment* function that recursively splits input text into a first word and a remainder, with the remainder then being treated as the input text. The computational inefficiencies of the recursion are handled by using the dynamic programming technique of MEMOIZING, or caching, the substrings in order to avoid having to recalculate their probabilities. In this way, the algorithm multiplies the probabilities for all possible segmentations together and finds the candidate with the highest probability to be the most likely overall segmentation. As an example, consider the *category* segmentation of the term *category*: *segment* would compare the joint likelihood of \(P(\text{cat})\) and \(P(\text{egory})\) and find it to be smaller than \(P(\text{category})\); it would thereby ultimately choose *category* as the most likely “segmentation”.

For a Python implementation of Norvig’s algorithm, please see Appendix A.

### 3.2 An Extension of the Existing Methodology

Thus far, I have outlined the problem space of a need to segment Twitter hashtags; in doing so, I have provided an overview of the workings and features of Twitter, sketched an existing word segmentation method, and argued for a hashtag-specific method of segmentation. It is here that I can plainly state the parameters of how this algorithm differs from the extant version. As previously mentioned, a possible method to handle false negatives – those terms in testing that should cross the threshold of lexicalization, but do not due to a lack of training data – is to orient our language model towards the domains that a
given test document resides in. In addition to the work mentioned above, the task of researching methods of being able to quickly adjust the model to accommodate data not seen in training, the subsequent implementation of one such method, and an examination of its efficacy together constitute the contribution this work offers. In the sections that follow, I discuss the parameters of this work, along with some technical and operational challenges. I explore two possible methods of online corpus updating and describe the integration of one such method into the existing non-hashtag-centric methodology.

3.3 OPERATIONAL ISSUES

Given the need to keep the training data for a given tweet relevant to that tweet, an obvious solution space involves some form of language model updating. However, as already noted, a wholesale training set update for every test document is infeasible. In tackling this problem, I begin by borrowing from topic-based language models the idea that since documents that share topics share similar term frequency distributions, different models should be trained for different topics (Gildea & Hofmann 1999). Topic-mixture models examine some history of a test document in order to determine how to weight a particular topic’s training set in order to best generate a custom language model; however, I suggest that the brevity of text in any given test tweet implies that the number of topics in that tweet is low (in fact, one) and that therefore mixing is not appropriate. This monographic characteristic of tweets leads suggests the hypothesis that terms found within a hashtag will also be seen in the text of documents related to that hashtag, given enough documents to sample from. If so, those documents could be a viable source of training data for segmenting the hashtag. The problem is twofold: how do we define this relationship, and how do we obtain such documents?

3.3.1 Web-Based Results: A Dead End

One way is to answer both questions is to view them as a generalized QUERY EXPANSION problem: if a hashtag itself is treated as a query, then terms associated with that hashtag may refine and extend the query to return more relevant results than the hashtag alone. More specifically, the set of non-closed-class words contained in the text of a tweet are treated as the PSEUDO-RELEVANT results immediately retrieved by a “query” containing
the hashtag; some combination of the members of these “results” are then sent as one or more queries to a WWW search engine to return training data.

I was initially attracted to this general technique due to the large amount of training data that was potentially available. Even better, through the use of a search engine API against a general query, it is often easy to return query-biased document summaries (Tombros & Sanderson 1998) that yields hyper-relevant information “for free”, as it were; i.e., the data found in the snippets of text included with results of a given query are presumably more directly relevant to that query than the whole documents that the snippets are taken from, due to prior work in document summarization by the search engine engineers. If this method is used, a frequency distribution over all terms found in the summaries (up to a given cutoff) generated from one or more queries containing one or more of the tweet “results” can be calculated. One then hopes that segment will correctly utilize this distribution.

However, several pragmatic factors conspire against the implementation of this technique. It assumes that an API is available to the extent needed. As of this writing, both Google (Google Developers 2012)\textsuperscript{17} and Bing (2012)\textsuperscript{18} have transitioned to paying services, which may not suit researchers. It also assumes that the search engine used updates its index quickly enough to include results relevant to the hashtag. Perhaps the most critical, however, is the reliability of the results.

Suppose some scheme had been instituted to query a search engine with a bigram. As Nakov and Hearst (2005) note:

Consider the bigrams $w_1w_4$, $w_2w_4$, and $w_3w_4$ and a page that contains each bigram exactly once. A search engine will contribute a page count of 1 for $w_4$ instead of a frequency of 3; thus the number of page hits for $w_4$ can be smaller than that for the sum of the bigrams that contain it. (2)

In this case, such concerns immediately throw out any possibility of modifying the search method to utilize simple hit counts. Even more important is the realization that search engine results are a black box with a number of variables affecting the results. Funahashi and

\textsuperscript{17} https://developers.google.com/custom-search/v1/overview Updated August 12\textsuperscript{th}, 2012 and retrieved January 2\textsuperscript{nd}, 2013.

\textsuperscript{18} http://datamarket.azure.com/dataset/bing/search Updated April 11\textsuperscript{th}, 2012 and retrieved January 2\textsuperscript{nd}, 2013.
Yamana (2010) attempted to gauge the reliability of hit counts in the face of the known phenomenon of the counts’ “dancing”, or changing between queries of the same term(s). This change occurs as the result of connecting to different servers between queries, any of which may hold a portion of an index that is not synchronized with other portions. Furthermore, the method and speed of index updating lead to variable periods of stability between the “dancing” phases. Funahashi and Yamana, examining the behavior of the Bing and Yahoo! search engines, concluded that the hit count for a given query could be deemed stable when it keeps almost the same number over the course of a week. The demands of gathering training data quickly for a currently viable topic indicate that this is a losing strategy. Generally speaking, due to the importance of having accurate representation of query results to generate a precise frequency distribution of terms, relying on web search to generate training data for this task does not seem currently viable.

### 3.3.2 Twitter-Based Results

Another source of training data that seems more accessible is the collection of tweets containing a given hashtag. This relationship is clearly defined and it is considerably easier to obtain the text of these tweets via the Twitter API than it is to obtain relevant text from the general Web with a search engine API. Although the Twitter API suffers in comparison to the sheer quantity of the potential data available through a web API, it more than acquits itself with the richness and relevance of its resources. The Twitter API exposes tweets that can be filtered by hashtag, location, language, and other criteria; moreover, using it to retrieve training data focuses this research. Specifically, we can formulate null and alternative research hypotheses:

- $H_0$: For a given non-segmented hashtag, there is no statistical difference between the performances of a segmenter using a static training corpus and a segmenter using a training corpus updated with terms found in tweets containing said hashtag.

- $H_a$: For a given non-segmented hashtag, a segmenter using a training corpus updated with terms found in tweets containing said hashtag outperforms a segmenter using a static training corpus to a statistically significant degree, when “outperforming” is measured as yielding a greater number of completely correctly segmented strings from the original hashtag’s concatenated string.
3.4 TECHNICAL ISSUES

The Twitter API is actually composed of two discrete API groups: REST and streaming. The former follows established REST procedures\(^{19}\) with a series of HTTP requests, while the latter issues long-lived HTTP connections over which data is continually sent. Both are subject to rate limiting, but differ in the types of resources offered. The streaming APIs issue tweets and user data; the REST API offers a variety of resources and actions, such as tweet retrieval, profile updating, and spam reporting (Twitter 2012c).\(^{20}\) As of version 1.1 of the REST API, rate limits over a 15 minute interval are capped at 15 or 180 GET requests, depending on the requested resource (Twitter n.d.a).\(^{21}\) The Public streaming API offers three endpoints by which to acquire tweets; only one of which, the “Firehose”, returns all public statuses. Access to the Firehose is generally withheld from all but a few large third-party partners (Twitter 2012a).\(^{22}\) The other two endpoints, “Sample” and “Filter”, are together sometimes referred to informally as the “Spritzer”, due to the relatively small amount of data they return (Twitter 2012d).\(^{23}\) Much of the academic research that has been performed on Twitter has utilized the “Spritzer” feed; see Bernstein and colleagues 2010, Hong and colleagues 2011, and Yang and Counts 2010, among others.

I accessed the Twitter API by several means. Rather than working directly with the API, I employed various Python libraries and other APIs that offered some amount of built-in error handling and other niceties. When acquiring REST resources, such as currently trending hashtags, I used the Python package twitter (Python 2014),\(^{24}\) which despite its name is not maintained nor officially sanctioned by Twitter. When retrieving streaming resources, I used

---

\(^{19}\) REST, or Representational State Transfer, is an HTTP-based standard of identifying and performing actions upon resources provided by a web API.

\(^{20}\) https://dev.twitter.com/docs/rate-limiting/1.1 Updated October 26\(^{\text{th}}\), 2012 and retrieved December 24\(^{\text{th}}\), 2012.


\(^{22}\) https://dev.twitter.com/docs/api/1.1/get/statuses/firehose Updated August 27\(^{\text{th}}\), 2012 and retrieved December 24\(^{\text{th}}\), 2012.

\(^{23}\) Twitter reserves the right to adjust the flow of data through the Spritzer; as of this writing it is currently downsampled to about 1% of the 100% access represented by the Firehose. See https://dev.twitter.com/discussions/2907, retrieved December 24\(^{\text{th}}\), 2012.

\(^{24}\) http://pypi.python.org/pypi/twitter/1.9.0 Retrieved December 24\(^{\text{th}}\), 2012.
Requests (Reitz 2014)\textsuperscript{25} to interact with Topsy (2014),\textsuperscript{26} an analytics service that exposes a powerful API capable of quickly retrieving a large amount of Twitter data.

### 3.5 Methodology

The data handling performed in this research can be roughly divided into three stages: the acquisition of hashtags that meet the testing criteria; the subsequent retrieval, transformation, and storage of the training text to be used in the segmentation of each hashtag; and the calculation of the segmentation results. However, as shown in Figure 3.1, these stages in practice are realized first as a continuous cycle of hashtag and text acquisition, with each iteration accompanied by its concomitant creation of frequency distributions for text associated with each hashtag. Upon completion of the data collection phase, the segmentations are then calculated.

#### 3.5.1 Hashtag Acquisition

A number of factors were involved in the selection of hashtags. In the initial design of this study, to ensure that sufficient training data were available for a given hashtag, the set of hashtags to be used was tentatively restricted to those that could be retrieved through the three methods of exploring trends that the Twitter REST API offers. Upon further inspection, I found that only the \textit{GET trends/place} (Twitter 2012b)\textsuperscript{27} method actually identifies trending topics; the other methods return locations that Twitter has trending topic information for, which are then used as input to \textit{GET trends/place}. (It is possible to programmatically retrieve currently trending hashtags (Tweitgeist n.d.)\textsuperscript{28} without using the API, but this was rejected as being outside the scope of the current study due to the amount of non-segmentation-oriented coding required.) This discovery proved to be fortuitous: since Twitter identifies locations through the use of Yahoo’s Where On Earth IDs (WOEIDs)(GeoPlanet n.d.),\textsuperscript{29} the ability to

\textsuperscript{25} http://docs.python-requests.org/ Retrieved December 24\textsuperscript{th}, 2012.
\textsuperscript{26} http://topsy.com/ Retrieved December 24\textsuperscript{th}, 2012.
\textsuperscript{27} https://dev.twitter.com/docs/api/1.1/get/trends/place Updated September 5\textsuperscript{th}, 2012 and retrieved January 8\textsuperscript{th}, 2013.
\textsuperscript{28} http://tweitgeist.colinsurprenant.com/ Retrieved November 29\textsuperscript{th}, 2012.
Figure 3.1. Overview of workflow.

Filter locations to those within the United States, as is afforded by the WOEID system, helped ensure that the text of most of the hashtags associated with said locations would be in English. (Although the Twitter API allows for language filtering, at the time I accessed it the results were not necessarily guaranteed.) Once the locations with potentially suitable trending topics were identified, the topics in all locations were filtered and reduced to a set of hashtags. Each iteration through the entire program, as further described below, generated a text file containing these hashtags as a separate collection. (Please see Appendix B for the relevant source code.)
3.5.2 Text Acquisition, Preparation, and Training

The next step in the program was individualized frequency distribution assembly. This required a number of sub-steps. As an overview, a search for sufficient training data for each hashtag was made, and if found for a given hashtag, that data set was converted to a hashtag-specific frequency distribution, building off the frequency distribution of terms pulled from Norvig’s unigrams file.

This workflow recognizes an inherent limitation of this research: due to the short lifespan of trending hashtags, there is a vanishing window of time in which to collect relevant text data for them. Yet given that the Twitter REST API institutes a rate limit on calls to it, a balance must be achieved between the number of hashtags collected in one iteration of the hashtag retrieval step and the amount of text data that may be transiently available for any given hashtag in the collection. The sample iteration mentioned above and in Section 2.2.2 took approximately 3 hours to run, and I used identical parameters in experimental iterations. This workflow did have the benefit of enforcing code modularity: get_hashtags.py, the Python implementation of this procedure (shown in Appendix B), is separated from get_text_data.py (shown in Appendix C).

Continuing in the sample iteration as supposed in Section 3.5.1, the program then submitted each hashtag in the current collection to Topsy, a search engine and metrics provider that indexes and analyzes a number of SNSs, most notably Twitter. This was primarily done in an effort to ensure data quality: the “Sample” endpoint for Twitter’s streaming API is subject to rate limits, and does not deliver matching Tweets beyond the streaming cap. Therefore, a popular hashtag that appears in tweets of sufficient number to sample is nonetheless subject to an earlier random sampling of unknown effect. However, as Topsy subscribes and allows secondary access to Twitter’s “Firehose”, this effect is avoided through the usage of their API. Another reason for using Topsy was the need to establish a minimum amount of text available for a given hashtag; by proxy, this leads to a check on some minimum number of tweets. This implied listening to the Twitter stream for

---

an unknown length of time until this constraint was satisfied. Conversely, Topsy’s API relays its Firehose access through its code base, which while slightly delaying access to the stream does provide valuable methods to act upon the data.

One such method yields the number of tweets containing a term. The program used this to quickly check for a sufficient number of tweets for each hashtag; if the check was passed, the text of tweets containing a given hashtag was retrieved in a JSON response format. All text collected for this given hashtag was subjected to NORMALIZING, the collective process of cleaning and standardizing data. This involved removing web links, special characters, and a very small number of stopwords that are artifacts of Twitter’s mechanics. In addition, all hashtags were removed in order to not contaminate the data. All usernames were removed, in compliance with IRB standards, so as to preserve anonymity. The end result was an unordered BOW (bag of words) collection for this and each hashtag in the current iteration.

With hashtags and their respective associated text in hand, the next step within the iteration was to create hashtag-specific frequency distributions. A frequency distribution created from Norvig’s unigrams file served as the basis for each hashtag-specific distribution. The unigrams file consisted of a simple tab-delimited mapping of terms to absolute frequency counts. A sample is shown below:

```
branch@Mímisbrunnr:~/code/segmenter/corpora$ head -10 unigrams.txt
the  23135851162
of   13151942776
and  12997637966
to   12136980858
a    9081174698
in   8469404971
for  5933321709
is   4705743816
on   3750423199
that 3400031103
branch@Mímisbrunnr:~/code/segmenter/corpora$ tail -10 unigrams.txt
googllr  12711
googlal  12711
googgoo  12711
googgol  12711
goofel   12711
```
Each hashtag, its BOW collection, and a copy of the reference unigrams frequency
distribution were then used to generate the hashtag frequency distribution.

Let \( N \) be the sum of the counts in the copy of the reference unigrams frequency
distribution, and let \( H \) be the total number of tokens in the BOW collection for a given
hashtag. Then consider some word \( w_1 \) that has count \( r \) in the copied frequency distribution
and count \( h \) in the hashtag BOW collection. Given the probability \( h/H \) of \( w_1 \) occurring in the
hashtag BOW collection, we expect to see \( w_1 \) occur \( h \ast \) times in a sample of size \( N \). We
conditionally replace count \( r \) with \( h \ast \) in the copied frequency distribution, where
\[
    h \ast = \frac{h}{H} \times N
\]
if \( h \ast > r \). In the case of \( r = 0 \) and \( h \geq 1 \), meaning that a word \( w_n \) occurred in the hashtag
BOW collection but not in the reference unigrams distribution, \( w_n \) was added to the copied
frequency distribution with count \( h \ast \). Arbitrarily, when \( r > h \ast \), \( r \) was not adjusted; this
reinforced the use of the values from the reference frequency distribution as a floor. The
values floor was necessary to ensure that the segmenter had a proper vocabulary to work
with. To take an extreme example, suppose that the tweet text collected for a particular
hashtag did not include the word the. Without the value of the in the reference frequency
distribution to rely on, the final hashtag distribution would not include the; then if the hashtag
in question did in fact include the, the probability of a correct segmentation could drop
almost to zero. Note that giving an unknown word a very small probability, as seen near the
end of Section 3.1, may help ensure correct segmentation, depending on the remaining
contents of the hashtag; however, as I note there, the normalization process just described is
intended to obviate the need to rely on that fallback.

After all normalization and updating was completed, the final hashtag frequency
distributions were then converted to text files in the format of the original unigrams file.
(Please refer to Appendix C for the Python code for this portion of the program.) At this
point, the iterative process would begin anew.
I typically retrieved approximately 50 United States-based WOEIDs per iteration. All iterations were given a maximum of 5 hashtags to retrieve from each location; earlier exploratory attempts to collect 10 hashtags per location resulted in a sufficiently long run time such that it became difficult to subsequently acquire a substantive amount of still-relevant text from the stream, as discussed at the beginning of Section 3.5.2. The hashtags objects were returned through the API in a JSON response format. As mentioned in Section 2.2.2, a sample iteration of such an attempt collected 91 distinct trends, 42 of which were hashtags. Each iteration of the hashtag acquisition script that was performed during the data sampling phase yielded an average of 11 hashtags after duplicates were removed.

Subsequent to acquiring the hashtag set, I submitted each member in the set as a query to Topsy’s searchcount method, which returns the count of results for a search query. If that count was found to exceed 500, that hashtag was deemed to have enough text available to train the updated segmenter. However, as a means of establishing an arbitrary performance floor for the segmenter, the query’s parameters specified that only 50 tweets were selected. Additionally, the query specified that results were limited to English and originally tweeted within the past day.

### 3.5.3 Data Treatment

Upon completion of hashtag and text acquisition, and the subsequent creation of hashtag-specific frequency distributions, the next step was to act on the data. In the sections that follow, I detail the procedure I followed to segment the hashtags.

#### 3.5.3.1 Database Setup

The database created was initially a simple single table with named attributes, as shown in Table 3.1.

This relation schema allowed me to track hashtags with unique identifiers. Although I took care in the hashtag acquisition stage of each iteration to eliminate hashtag duplication, this did not prevent popular and relatively-long lasting hashtags from persisting between

---

Table 3.1. Relation Schema

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute type</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID</td>
<td>Integer</td>
</tr>
<tr>
<td>instudy</td>
<td>Integer</td>
</tr>
<tr>
<td>text.original</td>
<td>String</td>
</tr>
<tr>
<td>text.seg.basic</td>
<td>String</td>
</tr>
<tr>
<td>score.seg.basic</td>
<td>Integer</td>
</tr>
<tr>
<td>text.seg.ext</td>
<td>String</td>
</tr>
<tr>
<td>score.seg.ext</td>
<td>Integer</td>
</tr>
</tbody>
</table>

iterations. As such, I found that reducing the 379 collected hashtags to a set yielded 136 unique terms. By adding a binary “instudy” field to the initial table in the database and populating it such that a group of identical hashtags would have only one instance marked “True”, I replicated that set. I then selected those “True” records and created a new working table from them.

3.5.3.2 SEGMENTATION

I ran a script that created the database and populated it with the new hashtags (see Appendix D) and then in quick succession ran both the original and the modified segmenters, both of which updated the database with their respective segmentations of the newly added hashtags.

As previously discussed, this work extends the basic segmentation algorithm outlined in Section 3.1. This includes not only the gathering of new and hopefully relevant corpora for hashtags to be segmented, but the actual segmentation procedure itself. A number of minor housekeeping alterations to Norvig’s code (again, see Appendix A) were necessary to implement the updated algorithm, mostly having to do with pointing the code at the correct corpus for any given hashtag being segmented. Please see Appendix E for a Python implementation of this updated segmenter, along with Appendix F for some general utility functions referenced throughout Appendices A – F.
3.5.4 Evaluation

The next step was manually segmenting the remaining hashtags to provide a gold standard to later compare the basic and extended segmenters’ results to. My approach in doing so was to rely only on my extant general knowledge in deciding segmentations, avoiding reliance on web searches to confirm or deny guesses. Although this method did introduce an element of bias, it did somewhat realistically emulate a human ceiling to the gold standard accuracy. Future research should involve not only a larger sample set, but more than one gold standard arbiter; specifically, having several arbiters would smooth out the differences in prior knowledge between the arbiters and allow for a broader and thus more defensible definition of “gold standard”.

There were several instances in which there was an entity referred to within a hashtag that I was not able to manually disambiguate due to a lack of domain knowledge. An example of this would be staystrongkidrauhl: the basic segmentation was stay strong kid rauhl, while the extended segmentation was stay strong kid rauhl. Although I suspected that the extended segmentation was correct, I did not recognize the reference and could not provide a gold standard of segmentation with my existing knowledge; thus, I threw that entry out, and others representative of the same case. At other times, I could only be reasonably certain of a referred entity, as in voteselena; in this case, the basic segmentation was votes elena but the extended segmentation was vote selena. Here, I was aided by the fact that simultaneously trending hashtags were vote1d and vote1duk, which I judged as referring to the boy band One Direction. Aided by this domain clue and having some prior knowledge of the individual, I concluded that selena referred to Selena Gomez, a singer and actress. Having recognized that name and being reasonably confident of the correct segmentation, I did not hold this or other similar cases out. Again, using a group of arbiters would have doubtless yielded varying interpretations of the correct segmentation of the likes of staystrongkidrauhl and voteselena, which could be taken into account when establishing the standard. Furthermore, the strict standard of pre-existing objective knowledge needed to validate correct segmentations that I employed is not necessarily motivated by the requirements of this study. Allowing the judge(s) to access external resources to confirm or deny beliefs would immediately allow a larger sample size.
Another minor issue in evaluation was the existence of initialisms. When a hashtag consisted solely of an unrecognized initialism, the segmenters usually did not split the string and I agreed with them in those cases; consequentially, if a segmenter did split it, I marked that segmentation as incorrect. When an unrecognized initialism occurred within a hashtag, I assessed the plausibility of various segmentations and scored accordingly when possible. For example, the hashtag `kcamessagetobieber` was interpreted by both segmenters as `kca message to bieber`, which I agreed with despite not recognizing `kca`; I therefore marked the segmentation as correct.

That both segmenters were able to properly extract `bieber` (as in pop singer Justin Bieber) may be surprising, but in actuality it reflects the algorithm’s inner workings. Despite the lack of updated training data, the baseline segmenter does not split `bieber` into sub-strings due to the simple fact that no valid English strings found in the original training data can be found within `bieber`. As a further example, both segmenters accurately split `operationmakebiebersmile` as `operation make bieber smile`. Contrast this situation with `voteselena: elena` does exist within the original training data, as does `votes`, and perhaps `selena` – but the baseline segmenter judged incorrectly that `votes elena` was the proper segmentation.

Given that the extended segmenter displays a facility for extracting unusual proper names from hashtags, it is not surprising that it is also adept at recognizing nicknames. A nickname that Justin Bieber’s fans have bestowed upon themselves is “beliebers”, a blend of “bieber” and “believers”. The hashtag `beliebershatepaparazzi` thus offers an interesting counterpoint to the earlier example of correct baseline segmentation of `bieber`. In this case, the baseline training data is able to split `beliebers` into the incorrect `be lie bers hate paparazzi`; however, the updated training data afforded to the extended segmenter yields `beliebers hate paparazzi`.

During evaluation, I also discovered an area in which the extended segmenter’s algorithm needs tuning. Due probably to the large amounts of typos in the extended segmenter’s training data, it was inconsistent in its term normalization. For example, while `I’ve` consistently becomes `ive` during basic segmentation, the extended segmenter would at times set it to `i ve`. Other contractions in the extended segmentations displayed similar behavior. Later research, particularly in the context of adjusting the parameters of the data
collection for the extended segmenter, may address this issue. Here again, as well, the issue of objectivity rears its head, albeit in a different form: without an agreed-upon prior standard of text normalization, arbitrary decisions on the correctness of segmentations seriously undermine the validity of the research. That said, for the purposes of this work, I chose to add a “text.badnorm” field to the database table; in the final evaluation I judged the segmenters based on strict accuracy within the confines of the observations made above, but also marked this field as “True” when potential normalization issues occurred. In Section 4.1, I present my findings both with and without these hashtags thrown out.

Finally, despite setting the parameters of the data collection so as to only collect English tweets, a few non-English hashtags (as evidenced by usage of non-ASCII characters) did make their way into the data set. These were thrown out as well. Having established a standard to evaluate against, I binomially classified each of the two computationally-generated segmentations within a given record against the gold standard segmentation and assigned a score.
CHAPTER 4

RESULTS AND DISCUSSION

In the pages that follow, I show that hindered by a small sample size and an underperforming algorithm, the extended segmenter does not fare better than the baseline segmenter; however, the results do appear to clearly point towards a means of improved performance.

4.1 PRESENTATION OF THE FINDINGS

After categorization, the sample size $n$ of hashtags that were evaluated was 132. I found that the basic segmenter was accurate in $X = 112$ instances overall; thus the baseline sample proportion is $p = \frac{x}{n} = \frac{112}{132} = 0.84$. Given $p$ and the formula for the maximum error of estimate value for $n$:

$$E = z_{\alpha/2} \frac{pq}{n}$$

and the formula for finding a confidence interval for a proportion:

$$p - (z_{\alpha/2}) \frac{pq}{n} < p < p + (z_{\alpha/2}) \frac{pq}{n}$$

where $q = 1 - p$ and $z_{\alpha/2} = 1.96$ (indicating a 95% confidence level), we are only able to establish the interval for the baseline proportion at $77.7\% < p < 90.2\%$. Should we want a more reasonable maximum error we must find a larger value to assign to $n$:

$$n' = pq \frac{z_{\alpha/2}^2}{E}$$

Assuming a maximum of $p = 0.5$ at a consistent confidence level of 95%, we see that we would need a minimum sample size $n' = 385$ hashtags. It is thus unfortunate that time constraints prevented further data collection, but at worst this work may viewed as a pilot study yielding a future starting $p$. In Section 4.2.1 I elaborate more on the parameters of possible future research, including sample size.
Given this, I present the results obtained in evaluation as indicative of the modified algorithm showing potential by slightly beating the baseline performance.

As shown in Figure 4.1, the extended segmenter showed an improvement in accuracy over the basic segmenter. Before the instances of bad normalization were thrown out, the baseline segmenter yielded an accuracy of 84.84% to the extended segmenter’s 87.87%, an improvement of 3.03% by the extended segmenter. After the bad normalization instances were discarded, the extended segmenter yielded an accuracy of 81.06%, an improvement of 5.31% over the baseline segmenter.

![Scores by result set and segmenter](image)

**Figure 4.1. Performance of segmenters on hashtags.**

The next question of course is whether these measurements represent a statistically significant improvement. In examining this, I chose to bypass the standard z test for comparing two proportions due to its requirement that the samples must be independent of each other. Instead, as the two segmenters performed on instances of the same given string and in doing so created a natural pairing of two results from that string, the proper method of comparison utilizes McNemar’s test (see Dietterich 1997). This test specifically assesses the
significance of two proportions that are correlated in some way, such as having matched-pair samples.

To begin, we construct a 2x2 contingency table (see Table 4.1) and fill it in with values from the initial result set that did not discard bad normalization instances:

Table 4.1. Contingency Table for McNemar’s Test on Evaluation Results (No Discards)

<table>
<thead>
<tr>
<th>Accuracy count of baseline segmenter</th>
<th>Success</th>
<th>Failure</th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>107</td>
<td>5</td>
<td>112</td>
</tr>
<tr>
<td>Failure</td>
<td>9</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>Totals</td>
<td>116</td>
<td>16</td>
<td>132</td>
</tr>
</tbody>
</table>

The basic intuition of this test is that the difference of the probabilities of the two proportions reduces conceptually to the difference between two cells: the intersection $i$ of baseline segmenter failures and extended segmenter successes, and the intersection $j$ of baseline segmenter successes and extended segmenter failures. We then apply a form of the chi-square test:

$$McNemar's \chi^2 = \frac{(i - j)^2}{i + j}$$

Given that $i = 9$ and $j = 5$, with $df = 1$ and a 95% significance level given as a critical value of 3.85, we conclude that since $\chi^2 = 1.14 < 3.85$, the overall performance of the extended segmenter does not indicate a statistically significant improvement over the overall performance of the baseline segmenter. The null hypothesis holds in this data frame.

The results given in Table 4.2 for the evaluation done on the set with instances having normalization issues thrown out, however, tell a different story. Performing the same calculations as shown earlier yields $\chi^2 = 7.0 > 3.85$; that is, when normalization considerations are included, the performance of the extended segmenter indicates a statistically significant improvement over the performance of the baseline segmenter. In this case, we cannot reject the alternative hypothesis.
Table 4.2. Contingency Table for McNemar’s Test on Evaluation Results (with Discards)

<table>
<thead>
<tr>
<th></th>
<th>Accuracy count of extended segmenter</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Success</td>
<td>Failure</td>
<td>Totals</td>
</tr>
<tr>
<td>Accuracy count of baseline segmenter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Success</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Failure</td>
<td>7</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>Totals</td>
<td>107</td>
<td>11</td>
<td>118</td>
</tr>
</tbody>
</table>

4.2 DISCUSSION

Given the somewhat inconclusive results seen in this study, I suggest that more research is needed. The numerous optimization possibilities available and the potential payoff of a reliable hashtag segmentation method indicate that such moderately promising results are sufficient to warrant further work.

4.2.1 Error Analysis

As discussed in Section 3.5.4, the extended segmenter is prone to making errors when faced with contracted tokens, such as *im* rather than *im* when either the segmenter has removed the contraction in *i’m* or it was not present in the hashtag to begin with. The fact that this error does not consistently occur points to issues in the training data retrieved rather than in the segmentation algorithm. When we examine the reference unigrams frequency distribution, we find:

```bash
brandon@Mímisbrunnr:~/code/segmenter/corpora$ awk '{total += $NF} END \ 
{print total}' unigrams.txt
588124220187
brandon@Mímisbrunnr:~/code/segmenter/corpora$ grep -w 'im' unigrams.txt
im 37166923
brandon@Mímisbrunnr:~/code/segmenter/corpora$ grep -w 'i' unigrams.txt
i 3086225277
brandon@Mímisbrunnr:~/code/segmenter/corpora$ grep -w 'm' unigrams.txt
m 341583838
...
brandon@Mímisbrunnr:~/code/segmenter/corpora$ echo '37166923/588124220187' \ | bc -l
```
Here, the product of the probabilities of $i$ and $m$ is less than the probability of $im$, and so the basic segmenter produces $im$ for the hashtag #imthattypeofpersonwho. Compare this to the probabilities generated by the custom frequency distribution for this hashtag, which the extended segmenter segments as $i m$ that type of person who:

```
brandon@Mímisbrunnr:~/code/segmenter/corpora/tweets/completed$ awk '{total += $NF} END {print total}' #imthattypeofpersonwho.txt
999212868779
```

```
brandon@Mímisbrunnr:~/code/segmenter/corpora/tweets/completed$ grep -w 'im' #imthattypeofpersonwho.txt
im 37166923
```

```
brandon@Mímisbrunnr:~/code/segmenter/corpora/tweets/completed$ grep -w 'i' #imthattypeofpersonwho.txt
i 11658850173
```

```
brandon@Mímisbrunnr:~/code/segmenter/corpora/tweets/completed$ grep -w 'm' #imthattypeofpersonwho.txt
m 9067994579
```

```
brandon@Mímisbrunnr:~/code/segmenter/corpora/tweets/completed$ echo \\
'(3086225277/588124220187)*(341583838/588124220187)' | bc -l
.0000304780240896303
```

```
brandon@Mímisbrunnr:~/code/segmenter/corpora/tweets/completed$ echo \\
'(37166923/999212868779)' | bc -l
.00003719620129133901
```

Here, we see that due to the variance in word frequencies found in #imthattypeofpersonwho, the extended segmenter decides that $i m$ is the most probable segmentation. However, when examining the extended segmenter’s results further, we find examples in which contractions were properly handled, such as #donttrustsomeone that being segmented as dont trust someone that.

Although it may be the case that some sufficiently large data set generated for a hashtag may allow the extended segmenter to reliably make correct segmentations in these
cases, the length of time involved in data collection and the wide range of grammars seen in tweet text together suggest that the segmenter should search for these special cases and automatically apply adjustments to them as needed.

4.2.2 Improvements for Future Research

Future work in this topic must, first and foremost, collect a larger sampling in order to achieve statistically significant results. Before doing so, however, there are a number of operational parameters that can and should be fine-tuned during the collection process. The WOEIDs used to filter hashtags by location returned at most 5 hashtags in a data collection iteration; this parameter may not be optimal. Additionally, the set of WOIEDs used generally included locations within the United States as a means of ensuring English data; pilot studies correlating a location’s propensity to issue hashtags that generate sufficient text data to be included in this type of research may be called for.

There were many parameters of the data collection for the text associated with hashtags that should also be re-visited. The number of tweets associated with a given hashtag and selected for text collection was 50; this could easily be increased. Likewise, the minimum count of hashtag-containing tweets that had to be met in order to allow text collection could also be experimented with. Finally, the timeframe of tweets that examined to begin with was limited in this study to the past day; several longer timeframes are available in the API used to collect data.

Outside of data collection, there is the obvious issue of needing to fine-tune a tokenization strategy for text that very frequently does not adhere to orthographic conventions. The fact that the performance of the algorithms swung so wildly upon the removal of such problematic hashtags indicates how crucially a viable application of this research will depend on standardized text.

Finally, given that the bottleneck in data collection was accessing a stream of data, collecting instances of a filtered selection, and then returning to the stream, there would be a significant speed-up if the algorithm and code were adjusted to allow parallel processing.
4.2.3 An Application of This Work

In January of 2013, the Twitter engineering blog published a post[^33] that overviewed a method that they had put in place in order to more effectively monetize search results. As Twitter is so focused on ephemeral topics, it is subject to sudden spikes in search topics. The challenge inherent in such a suddenly popular topic is training the system in regards to the domain of the topic and what any related topics might be, and to do it quickly enough to be able to return domain-relevant ads along with searches for that topic. An example that the blog post used was that of the hashtag “#bindersfullofwomen”, which surged in popularity during the second U.S. presidential debate of 2012 in response to candidate Mitt Romney’s use of that phrase. Without sufficient training, a search for this hashtag during this debate would most likely have resulted in advertisements for office products along with the search results, rather than a domain-appropriate advertisement such as a request for campaign donations by President Barack Obama.

This is nominally an unsupervised learning problem, which Twitter’s engineers elected to solve by interjecting rapidly-respondent human judges on a massive scale through the use of Mechanical Turk[^34], a marketplace for connecting remote workers willing to do a range of tasks with businesses or developers needing human labor. These judges quickly answer surveys about a trending topic that are intended to categorize it.

However, as the state of the art of unsupervised learning advances, it is entirely plausible that Twitter may at some point elect to remove the human element from this process and depend entirely on computational parsing of these topics. Many of these topics will be hashtags; not all of them will be composed of English words, ready to be segmented. The ability to segment hashtags in order to extract entities from them will be a crucial first step in automatically classifying that hashtag.

Here, then, is where an improved segmentation method such as what I have outlined displays its worth. If it is possible to correctly update a segmenter’s training data for a hashtag based on text in tweets containing that hashtag, it seems extremely plausible that


newly-relevant entities that would not be recognized otherwise could instead be an element of an entirely unsupervised learning process that allows Twitter to refine its ability to learn about what those entities represent.
REFERENCES


TWITTER. 2010. To trend or not to trend... Online: http://blog.twitter.com/2010/12/to-trend-or-not-to-trend.html. (20 December, 2012.)


TWITTER. 2012b. GET trends/place. Online: https://dev.twitter.com/docs/api/1.1/get/trends/place. (8 January, 2013.)


TWITTER. 2013. GET statuses/user_timeline. Online: https://dev.twitter.com/docs/api/1.1/get/statuses/user_timeline. (4 November, 2013.)


APPENDIX A

SEGBASE MODULE
# -*- coding: utf-8 -*-
#!/usr/bin/env python

""

**********
The MIT License (MIT)

Copyright (c) 2008-2009 Peter Norvig

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

**********
This module implements Peter Norvig's word segmenter and extends it slightly by allowing for various input methods. This file is meant to be run from the command line.

@author: Brandon Devine
@contact: brandon.devine@gmail.com
@since: 8:55:36 PM on Nov 25, 2012

```
import argparse, operator, sys, os, re, sqlite3, time, logging

log = time.strftime('./logs/%H:%M:%S %d %b %Y', time.localtime())+'.log'

formatter = logging.Formatter('%(asctime)s %(message)s', '%H:%M:%S %d %b %Y')
handler = logging.FileHandler(log)
handler.setFormatter(formatter)
logger = logging.getLogger()
logger.addHandler(handler)
logger.setLevel(logging.DEBUG)
```
class Pdist(dict):

    """A probability distribution estimated from counts in a datafile.""

    def __init__(self, data=[], N=None, unkfn=None):
        for key, count in data:
            self[key] = self.get(key, 0) + int(count)  # since this is being populated en masse, all vals are initially 0; hence the + int(count)
        self.N = float(N or sum(self.itervalues()))  # if N is not supplied, back off to the sum of all values in the dict instance.
        self.unkfn = unkfn or (lambda key, N: 1./N)  # if unkfn is not supplied, back off to a simple estimation of an unknown word

    def __call__(self, key):
        if key in self: return self[key]/self.N  # returns the simple MLE if the calling key is in the instance's dict
        else: return self.unkfn(key, self.N)  # if not, back off to whatever we decided the unknown estimation method is

    def get_unk_word_prob(self, key, N):
        """Estimates the probability of an unknown word.""
        return 10./(N * 10**len(key))  # seat-of-the-pants heuristic

    def get_datafile(self, name, sep='\t'):
        """Reads key,value pairs from a file.""
        for line in file(name):
line = line.rstrip('n')
yield line.split(sep)

N = 1024908267229

Pw = Pdist(get_datafile('corpora/unigrams.txt'), N, get_unk_word_prob)

def get_Pwords(words):
    """Returns the Naive Bayes probability of a sequence of words."""  #although really, there's not much Bayesian voodoo going on
    return get_product(Pw(w) for w in words)  #Pw can take w as arg because of defined __call__ magic method

def get_product(nums):
    """Returns the product of a sequence of numbers."""
    return reduce(operator.mul, nums, 1)  #ex: with nums = [2,3,4], ((1x2)x3)x4) = 24

def get_splits(text, L=20):
    """Returns a list of all possible (first, remaining) pairs, \n    len(first)<=L."""
    return [(text[:i+1], text[i+1:])
            for i in range(min(len(text), L))]
    #ex: with text = 'spark', [('s', 'park'), ('sp', 'ark'), ('spa', 'rk'), ('spar', 'k'), ('spark', '')]
def memoize(f):
    '''Memoizes function f.'''
    table = {}
    def fmemo(*args):
        if args not in table:
            table[args] = f(*args)
        return table[args]
    fmemo.memo = table
    return fmemo

@memoize
def get_segs(text):
    '''Returns one of a list of words that is the best segmentation of text.'''
    if not text: return []
    candidates = ([first]+get_segs(remaining) for first,remaining in get_splits(text))
    return max(candidates, key=get_Pwords)

def normalize(text):
    '''Normalizes (hashtag) text by removing hashes and setting to lowercase.'''
    text = text.lower()
text = re.sub('#', '', text)
return text

def set_segs(data):
    "Handles data coming in as different formats and outputs as needed."
    conn = sqlite3.connect('hashtags.db')
curs = conn.cursor()
    # handles strings only from -s and -f
    if type(data) is str:
data = normalize(data)
    segs = get_segs(data)
    segs = ' '.join(segs)
    return segs
    # handles database entries
else:
    uid = data[0]
    inp = data[1]
    ninp = normalize(inp)
    segs = get_segs(ninp)
    segs = ' '.join(segs)
curs.execute('UPDATE tblHashtags SET "text.seg.basic" = ? \
WHERE "UID" = ?, (segs, uid))
    conn.commit()
    return segs

p = argparse.ArgumentParser(description="segbase.py")
p.add_argument("-s", "--string")
p.add_argument("-f", "--infile")

args = p.parse_args()

def read_input(args):
    """Reads data source from CLI."""
    if args.string:
        yield args.string
    elif args.infile:
        with open(args.infile,'r') as f:
            for line in f:
                yield line.rstrip("\n")
    else:
        pathname = os.path.abspath("")
        if os.path.exists(pathname+'/hashtags.db') == False:
logging.info('Please ensure that hashtags.db is in the current directory.')
sys.exit(1)
else:
    conn = sqlite3.connect('hashtags.db')
curs = conn.cursor()
curs.execute('SELECT * FROM tblHashtags WHERE "text.seg.basic" IS NULL')
rows = curs.fetchall()
for r in rows:
    yield (r[0],str(r[2]))

def main():
    logging.info('Started segbase.py at '+
        time.strftime("%d %b %Y %H:%M:%S", time.localtime()))
    logging.info('Corpus size: %s', N)
    for line in read_input(args):
        inp = line  #Needs to be set to line only when testing -s
        #inp = line[1]
        logging.info('Input: %s', inp)
        output = set_segs(line)
logging.info('Output: %s', output)
logging.info('Done at ' + time.strftime("%d %b %Y %H:%M:%S", \
    time.localtime()))

if __name__ == '__main__':
    main()
APPENDIX B

GET_HASHTAGS MODULE
#!/usr/bin/env python

'''get_hashtags module

This module accesses the Twitter REST API and extracts hashtags into a file.

@author: Brandon Devine
@contact: brandon.devine@gmail.com
@since: 3:16:39 PM on Jan 8, 2013
'''

import twitter, time, logging

twitter_api = twitter.Twitter()

myfile = time.strftime("%H:%M:%S %d %b %Y", time.localtime())+'_hashtags.txt'

log = time.strftime('./logs/%H:%M:%S %d %b %Y', time.localtime())+'.log'

formatter = logging.Formatter('%(asctime)s %(message)s', '%H:%M:%S %d %b %Y')

handler = logging.FileHandler(log)

handler.setFormatter(formatter)

logger = logging.getLogger()

logger.setLevel(logging.DEBUG)

def get_woeids():
    """Retrieves the set of Yahoo WOEIDs extant to the United States."""

    logging.info('Retrieving woeids...')

    woeads = []

    try:
        avail = twitter_api.trends.available()
time.sleep(25)
for entry in avails:
    if entry['countryCode'] == 'US':
        woeids.append(entry['woeid'])
except twitter.api.TwitterHTTPError:
    pass
logging.info(str(len(woeids))+' woeids retrieved.')
return woeids

def get_hashtags(woeids, number):
    """Retrieves up to $number trending hashtags for each location represented by a WOEID."""
    logging.info('Retrieving hashtags...')
    hashtags = []
    hashtagcount = 0
    for woeid in woeids:
        try:
            trends = twitter_api.trends._(woeid)
            time.sleep(60)
        except twitter.api.TwitterHTTPError:
            break
        i=0
        while i< number:
            try:
                trend = trends()[0]['trends'][i]['name']
                time.sleep(25)
                if trend.startswith('#'):
                    hashtagcount += 1
                    logging.info(str(hashtagcount)+' of '+
                                    str(number*len(woeids))+\
                                    ' potential hashtags found...')
                    hashtags.append(trend.lower())
except twitter.api.TwitterHTTPError or urllib2.URLError:
    break
    i += 1
logging.info('Ceasing hashtag retrieval.')
return hashtags

def assemble_hashtags(hashtaglist):
    """Creates a file of unique hashtags based on input."""
    unique_hashtags = set(hashtaglist)
    logging.info('Creating hashtag set...')
    with open('corpora/hashtags/>'+myfile, 'a+') as f:
        for entry in unique_hashtags:
            if entry.startswith('#'):  #double-check
                print >> f, entry
    logging.info(str(len(unique_hashtags))+' unique hashtags retrieved.')

def handle_hashtags(numhashtags=5):
    """Gathers ye functions while ye may."""
    woeidlist = get_woeids()
    hashtags= get_hashtags(woeidlist, numhashtags)
    assemble_hashtags(hashtags)

def main():
    logging.info('Started get_hashtags.py at '+'\
    time.strftime("%d %b %Y %H:%M:%S", time.localtime()))
    handle_hashtags()
    logging.info('Done at '+'\
    time.strftime("%d %b %Y %H:%M:%S", time.localtime()))

if __name__ == '__main__':
    main()
APPENDIX C

GET_TEXT_DATA MODULE
#!/usr/bin/env python

# get_text_data module

This module imports the file created by get_hashtags.py and builds a gold standard frequency distribution of terms from Peter Norvig's modified unigrams.txt. It then filters the Twitter stream for each hashtag, and if a sufficient number of tweets containing it are found, those tweets are collected, cleaned, and compared to the gold standard in order to create a hashtag-specific frequency distribution.

@author: Brandon Devine
@contact: brandon.devine@gmail.com
@since: 5:10 PM on Jan 8, 2013

import time, re, glob, requests, json, copy, logging
from collections import defaultdict
from utilities import read_api_key
log = time.strftime('./logs/'+'%H:%M:%S %d %b %Y', time.localtime())+'.log'

formatter = logging.Formatter('%(asctime)s %(message)s', '%H:%M:%S %d %b %Y')
handler = logging.FileHandler(log)
handler.setFormatter(formatter)
logger = logging.getLogger()
logger.addHandler(handler)
logger.setLevel(logging.DEBUG)

def retrieve_hashtags(path='/home/brandon/code/segmenter/corpora/hashtags'):
    """Pulls hashtags out of the file(s) created by get_hashtags.py.""
    hashtags = []
    for hfile in glob.glob(path+'/*.txt'):
        with open(hfile, 'r') as f:
            for hashtag in f.readlines():
                hashtags.append(hashtag.strip())
    return hashtags

def retrieve_text(hashtag, numtweets=500):    #changing numtweets necessitates changing tweetpayload params below
    """""
Uses Topsy (because they're cool) API to search for tweets containing a given hashtag. If said hashtag occurs often enough, the tweet text is grabbed, sent to the cleaners, and added to a raw corpus that will inform a hashtag-specific frequency distribution.

```
payload = {'apikey': read_api_key('./topsyapikey.txt'), 'q': hashtag}

r = requests.get("http://otter.topsy.com/searchcount.json", \
    params=payload)

info = json.loads(json.dumps(r.json, sort_keys=True, indent=4))

ttl = [] # total term list of all text in numtweets tweets with hashtag

if info['response']['h'] < numtweets:
    return ttl

else:
    i = 1

    while i <= 5:
        try:
            # print 'Getting page ' + str(i) + '...' 
            tweetpayload = {'apikey': read_api_key('./topsyapikey.txt'), \
                'q': hashtag, 'allow_lang':'en', 'window':'h23', \
                'page':i, 'perpage':10}

            r = requests.get("http://otter.topsy.com/search.json", \
                params=tweetpayload)

            texts = r.json['results']['text']

            for text in texts:
                # process text

        except:
            pass
```

```
params=tweetpayload)
tweets = json.loads(json.dumps(r.json, sort_keys=True, indent=4))
['response']['list']
for tweet in tweets:
    text = json.loads(json.dumps(tweet, sort_keys=True, indent=4))
    ['content']
    text = clean_text(text)
    ttl.extend(text.split())
    # print 'Tweet added to term list...' 
except KeyError:
    pass
    i += 1
logging.info('Returning term list...')
return ttl

def clean_text(text):
    """Normalizes text to lowercase and removes usernames, links, extraneous characters, hashtags, and stopwords."""
    # fix links and strip extraneous characters
    text = re.sub(r'http://t.co/[\w]+', '', text)  # Twitter converts all links to its t.co domain
    text = re.sub(r'\[\]{}?!"\.;:/\[\]' ' ' text)
text = re.sub(\xe2\x80\x9c|\xe2\x80\x9d', ", text)  # lame left and right double-quotes
#delete usernames
text = re.sub('\s*@\s*(\w+)', r' ', text)
#standardize to lower
text = text.lower()
#remove "stopwords"
p = re.compile(('(#\S*)|rt|.\.\.\.')  #'...' is appended to overlong tweets
    text = p.sub('', text)
# get rid of crufty whitespace
    text = ' '.join(text.split())
return text

def get_unigram_corpus():
    """Translates the existing unigrams.txt corpus into dictionary-based
    frequency distribution."""
    freqdist_unigrams = defaultdict(int)
    with open('corpora/unigrams.txt', 'r') as f:
        for l in f.readlines():
            freqdist_unigrams[l.strip().split('t')[0]] = \
int(l.strip().split('t')[1])
    return freqdist_unigrams
def make_hashtag_corpus(hashtag, termlist, freqdist_unigrams):
    
    """Produces a hashtag-centric variant of unigrams.txt that weights the
terms associated with that hashtag."""
    
    # make a frequency distribution of terms associated with the supplied hashtag
    freqdist_hashtag = defaultdict(int)
    for term in termlist: freqdist_hashtag[term] += 1

    ratio = \sum(freqdist_unigrams.itervalues())/sum(freqdist_hashtag.itervalues())  # normalizing factor
    new = copy.deepcopy(freqdist_unigrams)    # updated dict that combines pertinent k,v from hashtags with the majority of unigrams

    for k in freqdist_hashtag.keys():
        # for tokens common to the unigrams and hashtag corpora:
        # if the count of a token from unigrams.txt is lower than the normalized count
        # of that token in the hashtag corpus, update new with normalized hashtag count
        if k in new.keys() and new[k] < int(round(ratio*freqdist_hashtag.get(k))):
            new[k] = int(round(ratio*freqdist_hashtag.get(k)))

    # for tokens in the new hashtag corpus that don't exist in the unigrams corpus,
    # assign the normalized value from the hashtag corpus
else:
    new[k] = int(round(ratio*freqdist_hashtag.get(k)))
#tokens found in unigrams but not hashtags already exist in new at their proper ratio.
#make the new corpus text file from the accumulated tokens and their counts in new
myfile = str(hashtag)+'.txt'
with open('corpora/tweets/'+myfile, 'w+') as f:
    for k,v in sorted(new.items(), key=lambda tup: tup[0]):
        try:
            print >> f, k+'\t'+str(v)
        except UnicodeEncodeError:
            pass

def main():
    logging.info('Started get_text_data.py at '+\
    time.strftime("%d %b %Y %H:%M:%S", time.localtime()))
    logging.info('Retrieving hashtags...')
    hashtaglist = retrieve_hashtags()
    hashtaglist = ['#SanDiego']  # use for one-off tests
    logging.info('Building gold standard corpus...')
    unigrams = get_unigram_corpus()
    logging.info('Building hashtag corpora...')
for hashtag in hashtaglist:
    logging.info('Retrieving text for '+str(hashtag)+' ...')
    termlist = retrieve_text(hashtag)
    if len(termlist) == 0:
        logging.info('Hashtag corpus discarded due to lack of data.')
    else:
        logging.info('Making corpus for '+str(hashtag)+' beginning at \n'+time.strftime("%d %b %Y %H:%M:%S", time.localtime()))
        make_hashtag_corpus(hashtag,termlist,unigrams)
    logging.info('Done at '+time.strftime("%d %b %Y %H:%M:%S", \n       time.localtime()))

if __name__ == '__main__':
    main()
APPENDIX D

INIT_DATABASE MODULE
init_database module

This module creates a database populated with hashtags imported from the files created by get_text_data.py.

@author: Brandon Devine
@contact: brandon.devine@gmail.com
@since: 5:10 PM on Jan 8, 2013

import sqlite3, time, os, logging

log = time.strftime('./logs/'+'%H:%M:%S %d %b %Y', time.localtime())+'.log'

formatter = logging.Formatter('%(asctime)s %(message)s', '%H:%M:%S %d %b %Y')
handler = logging.FileHandler(log)
handler.setFormatter(formatter)
logger = logging.getLogger()
logger.addHandler(handler)
logger.setLevel(logging.DEBUG)

def setup_db(dbname='hashtags.db'):
    """Creates a database for hashtags, their segmentations, and their respective scores."""
    pathname = os.path.abspath('')
    global conn
global curs
    #if there is no database set up in script directory yet:
if os.path.exists(pathname+'/'+dbname) == False:
    conn = sqlite3.connect(str(dbname))
    curs = conn.cursor()
    curs.execute("CREATE TABLE tblHashtags (UID INTEGER PRIMARY KEY, \
    "instudy" INTEGER DEFAULT 0, \
    "text.original" VARCHAR(42), \
    "text.seg.basic" VARCHAR(42), \
    "score.seg.basic" INTEGER DEFAULT 0, \
    "text.seg.ext" VARCHAR(42), \
    "score.seg.ext" INTEGER DEFAULT 0)"")
    conn.commit()
# else re-initialize the existing database
else:
    conn = sqlite3.connect(str(dbname))
    curs = conn.cursor()

def populate_db(path='/home/brandon/code/segmenter/corpora/tweets'):
    """Pulls hashtags out of the files created by get_text_data.py.""
    for f in os.listdir(path):
        if f.endswith('.txt'):
            curs.execute("INSERT INTO tblHashtags (UID, 'text.original') VALUES (null, ?)"",(f.replace('.txt',''),))
    conn.commit()

def main():
    logging.info('Started init_database.py at '+time.strftime("%d %b %Y %H:%M:%S", time.localtime()))
    logging.info('Initializing database...')
    setup_db()
    logging.info('Populating database...')
    populate_db()
    logging.info('Done at '+time.strftime("%d %b %Y %H:%M:%S", ")
time.localtime())

if __name__ == '__main__':
    main()
APPENDIX E

SEGE XT MODULE
The MIT License (MIT)

Copyright (c) 2008-2009 Peter Norvig

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

segext module

This module implements Peter Norvig's word segmenter and extends it by providing
a method by which the corpus dependency can be updated so as to allow for
segmentation of text that may be relevant to the current timeframe.

@author: Brandon Devine
@contact: brandon.devine@gmail.com
@since: 8:55:36 PM on Aug 25, 2012

import argparse, operator, re, sqlite3, os, sys, time, logging
from collections import defaultdict

log = time.strftime('./logs/'+'%H:%M:%S %d %b %Y', time.localtime())+'.log'

formatter = logging.Formatter('%(asctime)s %(message)s', '%H:%M:%S %d %b %Y')
handler = logging.FileHandler(log)
handler.setFormatter(formatter)
logger = logging.getLogger()
logger.addHandler(handler)
logger.setLevel(logging.DEBUG)

class Pdist(dict):
    """A probability distribution estimated from counts in a datafile.""
    def __init__(self, data=[], N=102490826729, unkfn=None):
        for key,count in data:
            self[key] = self.get(key, 0) + int(count)
        self.N = float(N or sum(self.itervalues()))
        self.unkfn = unkfn or (lambda key, N: 1./N)
    #an already-created instance of Pdist, when called, executes __call__.
    #the instance is callable like a function (meaning that Pw below can take args).
    def __call__(self, key):
        if key in self: return self[key]/self.N
        else: return self.unkfn(key, self.N)

def get_unk_word_prob(key, N):
    return 10./(N * 10**len(key))

def get_datafile(name, sep='\t'):
    """Reads key,value pairs from file.""
    for line in file(name):
        line = line.rstrip('\n')
        yield line.split(sep)
def get_Pwords(words):
    """The Naive Bayes probability of a sequence of words.""
    return get_product(Pw(w) for w in words)

def get_product(nums):
    """Returns the product of a sequence of numbers.""
    return reduce(operator.mul, nums, 1)

def get_splits(text, L=20):
    """Returns a list of all possible (first, remaining) pairs, \n    len(first)<=L."
    return [(text[:i+1], text[i+1:])
            for i in range(min(len(text), L))]

def memo(f):
    """Memoizes function f.""
    table = {}
    def fmemo(*args):
        if args not in table:
            table[args] = f(*args)
            return table[args]
    fmemo.memo = table
    return fmemo

@memo
def get_segs(text):
    """Returns one of a list of words that is the best segmentation of text.""
    if not text: return []
    candidates = ([first]+get_segs(remaining) for \
               first,remaining in get_splits(text))
    return max(candidates, key=get_Pwords)
def normalize(text):
    '''Normalizes (hashtag) text by removing hashes and setting to lowercase.'''
    text = text.lower()
    text = re.sub('#', '', text)
    return text

def set_segs(data):
    '''Handles data coming in as different formats and outputs as needed.'''
    try:
        conn = sqlite3.connect('hashtags.db')
        curs = conn.cursor()
        curs.execute('SELECT * FROM tblHashtags WHERE "text.original" = ? AND "text.seg.ext" IS NULL', (unicode(data),))
        for row in curs:
            uid = row[0]
            inp = row[2]
            ninp = normalize(inp)
            segs = get_segs(ninp)
            data = ' '.join(segs)
            curs.execute('UPDATE tblHashtags SET "text.seg.ext" = ? WHERE "UID" = ?', (data, uid))
        conn.commit()
    except:
        inp = data
        ninp = normalize(inp)
        segs = get_segs(ninp)
        data = ' '.join(segs)
        return data

def get_corpus_counts(corpus):
    '''Translates the given corpus into a dictionary-based frequency
distribution to return the total count of tokens in the corpus"

freqdist = defaultdict(int)
with open(corpus, 'r') as f:
    for l in f.readlines():
        freqdist[l.strip().split('	')[0]] = \
        int(l.strip().split('	')[1])
return sum(freqdist.itervalues())

p = argparse.ArgumentParser(description="segext.py")
p.add_argument("-s", "--string")
p.add_argument("-f", "--infile")

args = p.parse_args()

def read_input(args):
    "]""Reads data source from CLI.""
    if args.string:
        yield args.string
    elif args.infile:
        with open(args.infile,'r') as f:
            for line in f:
                yield line.rstrip("\n")
    else:
        pathname = os.path.abspath"
    if os.path.exists(pathname+'hashtags.db') == False:
        logging.info('Please ensure that hashtags.db is in the \ 
        current directory."
        sys.exit(1)
    else:
        conn = sqlite3.connect('hashtags.db')
        curs = conn.cursor()
        curs.execute('SELECT * FROM tblHashtags WHERE \

"text.seg.ext" IS NULL')
    rows = curs.fetchall()
    for r in rows:
        yield (r[2])

def main():
    logging.info('Started segext.py at ' + \
                 time.strftime("%d %b %Y %H:%M:%S", time.localtime()))
    for line in read_input(args):
        try:
            pathname = os.path.abspath('')
            logging.info('Input: %s', str(line))
            N = get_corpus_counts(pathname + '/corpora/tweets/' + str(line) + '.txt')
            logging.info('Corpus size: %s', str(N))
            global Pw
            Pw = Pdist(get_datafile(pathname + '/corpora/tweets/' + str(line) + \
                           '.txt'), N, get_unk_word_prob)
            output = set_segs(line)
            logging.info('Output: %s', str(output))
        except IOError:
            pass
    logging.info('Done at ' + time.strftime("%d %b %Y %H:%M:%S", \
                                           time.localtime()))

if __name__ == '__main__':
    main()
APPENDIX F

UTILITIES MODULE
utilities module

This module contains a few housekeeping functions.

@author: Brandon Devine
@contact: brandon.devine@gmail.com
@since: 8:42:57 PM on Dec 15, 2012

import os, operator, signal, errno, time
from functools import wraps

def read_api_key(keyfile):
    """Reads API key from file."""
    with open(keyfile, 'r') as f:
        return f.readline().strip()

def datafile(name, sep='\t'):
    """Read key,value pairs from file."""
    for line in file(name):
        line = line.rstrip('n')
        yield line.split(sep)

def read_login_file():
    pathname = os.path.abspath('')
    try:
        if os.path.exists(pathname+'/'+login.txt'):
            pathname = pathname+'/'+login.txt'
            f = open(pathname)
return f.readline().strip(), f.readline().strip()
f.close()
except:
    print 'The file login.txt was not found. Please refer to twitterinfo.py.'

class TimeoutError(Exception):
    pass

def timeout(seconds=10, error_message=os.strerror(errno.ETIME)):
    def decorator(func):
        def handletimeout(signum, frame):
            raise TimeoutError(error_message)
        def wrapper(*args, **kwargs):
            signal.signal(signal.SIGALRM, handletimeout)
            signal.alarm(seconds)
            try:
                result = func(*args, **kwargs)
            finally:
                signal.alarm(0)
            return result
        return wraps(func)(wrapper)
    return decorator

def get_timing(func):
    def wrapper(*arg):
        t1 = time.time()
        res = func(*arg)
        t2 = time.time()
        print '%s took %0.3f ms' % (func.func_name, (t2-t1)*1000.0)
return res
return wrapper

def product(nums):
    """Return the product of a sequence of numbers.""
    return reduce(operator.mul, nums, 1)  # ex: with nums = [2,3,4], (((1x2)x3)x4) = 24