

RICE UNIVERSITY

Comparing vector-based and ACT-R memory models using large-scale datasets:  
User-customized hashtag and tag prediction on Twitter and StackOverflow

by

Clayton Stanley

A THESIS SUBMITTED  
IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE:



Michael D. Byrne, Chair  
Professor, Psychology and Computer Science



Philip Kortum  
Assistant Professor, Psychology



Devika Subramanian  
Professor, Computer Science and Electrical  
Engineering

Houston, TEXAS  
December 2014

## Abstract

Comparing vector-based and ACT-R memory models using large-scale datasets:

User-customized hashtag and tag prediction on Twitter and StackOverflow

by

Clayton Stanley

The growth of social media and user-created content on online sites provides unique opportunities to study models of declarative memory. The tasks of choosing a hashtag for a tweet and tagging a post on StackOverflow were framed as declarative memory retrieval problems. Two state-of-the-art cognitively-plausible declarative memory models were evaluated on how accurately they predict a user's chosen tags: an ACT-R based Bayesian model and a random permutation vector-based model. Millions of posts and tweets were collected, and both declarative memory models were used to predict Twitter hashtags and StackOverflow tags. The results show that past user behavior of tag use is a strong predictor of future behavior. Furthermore, past behavior was successfully incorporated into the random permutation model that previously used only context. Also, ACT-R's attentional weight term was linked to a common entropy-weighting natural language processing method used to attenuate low-predictor words. Word order was not found to be strong predictor of tag use, and the random permutation model performed comparably to the Bayesian model without including word order. This shows that the strength of the random permutation model is not in the ability to represent word order, but rather in the way in which context information is successfully compressed. Finally, model accuracy was moderate to high for the tasks, which supports the theory that choosing tags on StackOverflow and Twitter is primarily a declarative memory retrieval process. The results

of the large-scale exploration show how the architecture of the two memory models can be modified to significantly improve accuracy, and may suggest task-independent general modifications that can help improve model fit to human data in a much wider range of domains.

## Acknowledgements

I would first like to thank my advisor, Mike Byrne, for all of his feedback and suggestions for this dissertation research and also his excellent guidance throughout my time as a graduate student. I would also like to thank my parents, Mike and Patsy, for the positive influence that they have provided me throughout my life. Finally, I would like to thank my brothers Brendan and Justin, sister Kellen, extended family, and friends, for all of their support during my time in the Rice Psychology graduate program.

This work was supported in part by the Gertrude Maurin Fund. The resources were used to purchase high-performance computing hardware and the research would not have been possible without this crucial equipment.

I would also like to thank the creators of a few open-sourced software packages that were heavily leveraged to do this research: The majority of the modeling and analysis was done with the R statistical computing environment (R Core Team, 2014) and two important R packages: `data.table` (Dowle, Short, Lianoglou, with contributions from R Saporta, & Antonyan, 2014) was leveraged to implement the models efficiently, and `ggplot2` (Wickham, 2009) was used to produce the visualizations.

## Table of Contents

Abstract	ii
Acknowledgements	iv
List of Tables	vii
List of Figures	viii
<b>Chapter 1: Introduction</b>	<b>1</b>
The Task: Predicting User-Generated Tags . . . . .	3
Research Questions . . . . .	4
Roadmap . . . . .	9
<b>Chapter 2: Theoretical Background</b>	<b>10</b>
ACT-R Declarative Memory Theory . . . . .	10
Latent Semantic Analysis Memory Theory . . . . .	15
Vector-Based Memory Systems . . . . .	18
Comparison of Vector-Based Models and ACT-R . . . . .	25
Recommendation Systems . . . . .	28
Hashtag Prediction . . . . .	30
<b>Chapter 3: General Methodology</b>	<b>41</b>
Models . . . . .	41
Datasets . . . . .	41
Acquiring Datasets . . . . .	46
Tokenization . . . . .	46
Stemming . . . . .	47
Handling Stop Words . . . . .	47

<b>Chapter 4: Past User Behavior</b>	<b>49</b>
Base-level ACT-R model . . . . .	49
Method . . . . .	50
Results . . . . .	51
Discussion . . . . .	61
<b>Chapter 5: Combining Predictors</b>	<b>63</b>
Overall Method . . . . .	64
Adding Offset . . . . .	68
Twitter Subsets . . . . .	71
Stop Words . . . . .	73
Corpus Size . . . . .	82
Amount of Compression for Random Permutation . . . . .	87
Combining Prior and Context . . . . .	90
Word Order . . . . .	93
Coefficients for Predictors . . . . .	97
<b>Chapter 6: Combining Predictors on the Popular-Users Dataset</b>	<b>101</b>
Impact of Prior vs. Context . . . . .	101
User-customized Co-occurrence Matrix . . . . .	105
<b>Chapter 7: General Discussion</b>	<b>111</b>
Overall Results . . . . .	111
Implications and Future Directions . . . . .	118
<b>References</b>	<b>121</b>
<b>Appendix 1: Support Code</b>	<b>126</b>

## List of Tables

1	ACT-R declarative memory model . . . . .	11
2	Random permutation model . . . . .	22
3	Stanley and Byrne’s StackOverflow tag prediction model . . . . .	31
4	Base-level component of ACT-R declarative memory . . . . .	49
5	Summary of accuracy for each model and dataset. The Bayesian model includes entropy weighting for the context words, the RP model includes entropy weighting and log odds transformation when combining the context and prior components. Both models use a user-customized co-occurrence matrix for the Twitter popular-users dataset, since a global co-occurrence matrix is not available for this dataset. . . . .	111

## List of Figures

1	Example tweets on twitter.com . . . . .	5
2	Example post on stackoverflow.com . . . . .	6
3	Example of tag suggestions when creating a post on the SuperUser site . . .	33
4	Example post on Google+ . . . . .	34
5	Log-log plot of tag frequency versus rank for the StackOverflow dataset and Twitter popular-users and popular-hashtags dataset subsets. Rank was assigned to each unique tag after ordering by tag frequency (most frequent given smallest rank). . . . .	45
6	Tagging profile for a single user. The x-axis represents the specific hashtag a user chooses for a post and time progresses vertically. A point in the plot represents when a user chooses a specific hashtag for a specific post. A user's chosen hashtag for their oldest post that contains a hashtag is the lowest point on the plot. . . . .	52
7	Model tagging profile of a single user on StackOverflow . . . . .	53
8	Model tagging profile of a single user on Twitter . . . . .	54
9	Model performance for a single dataset slice for StackOverflow. This includes model accuracy as a function of decay rate value for each user in the dataset slice. Model accuracy is shown both when the standard form of base-level activation ( $B_i$ ) and the optimized learning form of the equation are used. . .	55
10	Model performance for a single dataset slice for Twitter. This includes model accuracy as a function of decay rate value for each user in the dataset slice.	55
11	Best-fit decay rate for a single dataset slice for StackOverflow. The histogram represents the count of users for each best-fit decay rate value. Each bar represents a discrete decay rate value that was explored. . . . .	57

12	Best-fit decay rate for a single dataset slice for Twitter. Best-fit values were slightly higher for Twitter, so the range of the histogram is higher for Twitter than StackOverflow. . . . .	57
13	Overall best-fit decay rate for StackOverflow and Twitter. Results are shown for the two Twitter and StackOverflow (SO) popular-users datasets. Error bars represent the 95% bootstrapped confidence interval for the median. . . . .	59
14	Overall model accuracy for StackOverflow and Twitter at optimal decay rate values. All error bars are the 95% bootstrapped confidence interval for the mean. . . . .	60
15	Impact of adding offset component for StackOverflow. Error bars represent the 95% bootstrapped confidence interval for the mean model accuracy across all runs in dataset. The random permutation models are represented as “RP” and Bayesian models as “Bayes.” . . . . .	69
16	Impact of adding offset component for Twitter. All error bars are the 95% bootstrapped confidence interval for the mean. . . . .	70
17	Model accuracy for each of the four Twitter popular-hashtags subsets . . . . .	73
18	Variance in observation counts of words for each tag for StackOverflow. Plots are shown for the four different methods for handling stop words. Each plot contains the standard deviation of the total counts for each word in context as a function of hashtag. High values mean that there are context words for a hashtag that have a high number of counts relative to the counts for all other context words for that hashtag. . . . .	75
19	Variance in observation counts of words for each tag for Twitter. . . . .	76

20	Stop-word techniques for the random permutation model for StackOverflow. The five methods for handling stop words are shown both for the title model component in isolation and the full model. “Stoplist”, “freq”, “entropy” are the stoplist removal method, removal based on frequency count, and weighting based entropy metric. . . . .	77
21	Stop-word techniques for the random permutation model for Twitter. The five methods for handling stop words are shown both for the context model component (i.e., no prior term) and the full model. . . . .	78
22	Stop-word techniques for the Bayesian model for StackOverflow . . . . .	80
23	Stop-word techniques for the Bayesian model for Twitter . . . . .	81
24	Effect of size of co-occurrence matrix for StackOverflow. Error bars represent the 95% bootstrapped confidence interval for the mean model accuracy across all runs in dataset. . . . .	83
25	Effect of size of co-occurrence matrix for Twitter. Error bars represent the 95% bootstrapped confidence interval for the mean. . . . .	85
26	Effect of compression on random permutation model for StackOverflow. Error bars represent the 95% bootstrapped confidence interval for the mean. . . .	88
27	Effect of compression on random permutation model for Twitter. Error bars represent the 95% bootstrapped confidence interval for the mean. . . . .	89
28	Random permutation model accuracy when using log-odds transformation on context for StackOverflow . . . . .	92
29	Random permutation model accuracy when using log-odds transformation on context for Twitter . . . . .	93
30	Model accuracy for various word order methods for the random permutation model for Twitter . . . . .	96

31	Coefficient values for each component of the full models for StackOverflow. Models used are the best-performing Bayesian and random permutation models (i.e., Bayesian and RP have entropy weighting, RP converts context activation from correlation to log odds). Error bars represent the standard deviation of coefficient values for each component. The standard deviation is used instead of the confidence interval of the mean so that the plots show the overall stability of each coefficient across the different runs. . . . .	99
32	Coefficient values for each component of the full models for Twitter . . . . .	100
33	Model accuracy when prior is removed for StackOverflow randomly-sampled dataset . . . . .	102
34	Model accuracy when prior is removed for Twitter popular-hashtags dataset	102
35	Model accuracy when prior is removed for StackOverflow popular-users dataset	104
36	Model accuracy with user-customized context component for StackOverflow. A model name with “user sji” uses a co-occurrence matrix customized for each user instead of a single global co-occurrence matrix. . . . .	107
37	Model accuracy with user-customized context component for Twitter . . . . .	108

## Chapter 1: Introduction

The ACT-R cognitive architecture (Anderson, 2007) contains one of the leading theories of human cognitive processing. The declarative memory component in ACT-R is a Bayesian statistical model. This is the same type of co-occurrence-based Naive Bayes statistical model that is prominent in the more general machine learning field. When these Bayesian models are used for machine learning, it is common to use a large number of observations to construct the co-occurrence matrix that is the foundation for the model. The primary reason for this is that the stability of statistical models increase as sample size increases.

However, it is much less common in the cognitive modeling community to use a large number of observations to build up declarative memory when creating ACT-R models. If a cognitive modeler is using the declarative memory component of ACT-R, it is common to either construct the co-occurrences by hand, or use a small training set of data to initially build up the declarative memory system. This approach can and has worked well. Essentially, the modeler is only including the observations in declarative memory that are relevant to the specific task that they are studying.

However, this approach is problematic for two main reasons: First, it means that the declarative memory system for every developed model must be hand crafted and customized for that model, which takes enormous time and effort. Second, it means that modelers are continuing to use small sample sizes for the declarative memory components in their models.

This has important implications for understanding declarative memory: First, it is difficult to verify and explore modifications to the declarative memory architecture because the datasets are not large enough or rich enough to rigorously test modifications to the theory. It also means that we do not yet fully understand how well the ACT-R declarative memory theory scales when the number of chunks begins to approach the number of chunks that are actually stored in human declarative memory.

There is, however, an opportunity to change this approach. With the massive growth of human-created content on social media sites (e.g., Twitter, StackOverflow, Wikipedia), and public access to many of those databases, it is now possible to evaluate psychological theories of declarative memory on large-scale real-world tasks where the theory can be tested on hundreds of millions to billions of data points. This will provide a way to test if these models scale, and evaluate the psychological validity of these models as the number of observations used increases by orders of magnitude.

A few researchers have started to use these large databases to evaluate how well declarative memory models scale, for both ACT-R Bayesian models (Douglass & Myers, 2010; Fu & Pirolli, 2007; Pirolli & Fu, 2003; Stanley & Byrne, 2013) and vector-based memory models (Jones & Mewhort, 2007; Recchia, Jones, Sahlgren, & Kanerva, 2010; Rutledge-Taylor & West, 2008; Sahlgren, Holst, & Kanerva, 2008).

The work in this study expands on previous work that has tested retrieval models on large-scale datasets. It focuses specifically on how two state-of-the-art declarative memory retrieval models (ACT-R Bayesian and vector-based) can be improved by testing these models on two real-world tagging tasks. One primary objective is to compare the random permutation vector-based model to ACT-R's declarative memory theory when large datasets are used for each model. Another objective is to improve each model, and measure how incorporating the strengths of each model into the other (i.e., modifying model components) influences accuracy. For example, it is currently unexplored how a vector-based model can be modified to retrieve results that are based not only on context, but also on the prior odds that a particular item in memory will be needed again. Large-scale datasets were used throughout the model development and evaluation process. This provides an ideal testing environment for constraints and assumptions for each theory to be rapidly tested and new model components to be hypothesized and quickly evaluated.

## The Task: Predicting User-Generated Tags

Social media sites such as Twitter, Facebook, StackOverflow, Google+, and Yelp are composed almost entirely of human-created content. The amount of user content on these sites is quite large, and continues to grow. Users of Twitter's microblogging service, for example, generate half a billion tweets per day (Twitter, 2012). However, a single user of a social media site is most likely interested in only a small fragment of this large amount of information. One general question to support the user in this domain is: How can we quickly and effectively connect users to the content that they care about?

A growing number of users on social media sites are focusing on and looking for recently-created information when performing a search (Jansen, Liu, Weaver, Campbell, & Gregg, 2011). In other words, users on social media sites want the fresh *stream* of information about a particular topic. This can certainly be seen in the main user interface views that these social media sites provide: Twitter's homepage showing followees' tweets, Facebook's homepage showing friends' posts, and StackOverflow's daily digest of posts for particular tags.

So how can we ensure that users are connected to the information streams that they care most about? One promising approach is to model and predict the user's goals when using and searching these social media sites (Rose & Levinson, 2004). If we have a better understanding of a user's goals, then we can tailor the information streams to the content that is most in line with these goals. For example, Twitter might suggest other users and hashtags for the user to follow, Facebook could suggest pages to like, Yelp could recommend businesses to check out. Researchers are currently looking for ways to automate the goal-identification task for more general search queries like those generated for Google (Jansen, Booth, & Spink, 2008; Lee, Liu, & Cho, 2005).

However, social media sites have an advantage over traditional search engines for identifying user goals: human-created content. Twitter has more than just the tweet that a user is currently composing to try and identify that user's goals. The site also has all of the

user's previous content, and can use that information to provide a much better prediction of a user's goals. Further, sites like Twitter, StackOverflow, Google+, and Facebook support ways for the user to explicitly identify their precise goals when creating content: hashtags. These user-created hashtags are a key indicator of a user's goals on a social media site.

These hashtags relate to a user's goals and interests because they are a form of human-based document tagging (Chang, 2010). Further, people are using hashtags as a way to create information streams on social media sites (Kwak, Lee, Park, & Moon, 2010). Twitter users, for example, commonly create and use hashtags for upcoming political events that interest them, such as debates and races (Diakopoulos & Shamma, 2010). So if we can predict what hashtags a user is interested in, we have a better understanding of the information streams that they care about. We can use that information to correctly suggest hashtags that they have previously used when they are generating new content.

Further, if a particular information stream has multiple hashtags associated with it (common for Twitter), these semantically similar hashtags can also be recommended to users. So we can tailor the system to suggest hashtags that the user may not yet know about but are of likely interest to them and growing in popularity across the site. We can help aid in their discovery of fresh and relevant information that matches their goals on the site.

## **Research Questions**

The core research question motivating this proposal is: What kinds of cognitively-plausible models can best predict the tags that a user on a social media site uses? This question can be tested by generating a prediction for the most likely used tag whenever a user is about to generate a tag, and then comparing the user's chosen tag to the model's prediction. This process can be framed as a memory retrieval problem. The process of suggesting a relevant tag to a user can be thought of as a memory retrieval request for a tag, given prior tag use and current context. That is, each user on a social

media site has created content that contains a set of tags within the context of each post.

Co-occurrence-based modeling has been shown as a potentially useful approach when predicting Twitter hashtag use (Efron, 2010) and StackOverflow tag use (Stanley & Byrne, 2013). Several memory retrieval models are based on this broad methodology, where a count is maintained of the times each contextual element (such as a word in a post) co-occur with a tag. At least two of the current memory retrieval models are based on a co-occurrence methodology: ACT-R's declarative memory (DM) retrieval theory and random permutation vector-based memory systems.

**StackOverflow and Twitter.** These two memory systems were compared across two domains where users generate tags for posts: Twitter and StackOverflow. Twitter is a microblogging service where users create 140 character “tweets” and broadcast those messages out to the users who follow their posts. A user is free to annotate important concepts or words in the tweet by preceding words with “#,” thereby making the word into a hashtag. After the tweet is created the hashtags become hyperlinks, and if clicked on will take a user to a summary feed of all of the tweets across Twitter that have used that hashtag. So these user-created hashtags on Twitter help connect the tweet to other tweets about the same topic through common hashtag use. Some recent and often used hashtags for Twitter are *#photography*, *#startup*, *#4change*, *#android*, and *#solar*.

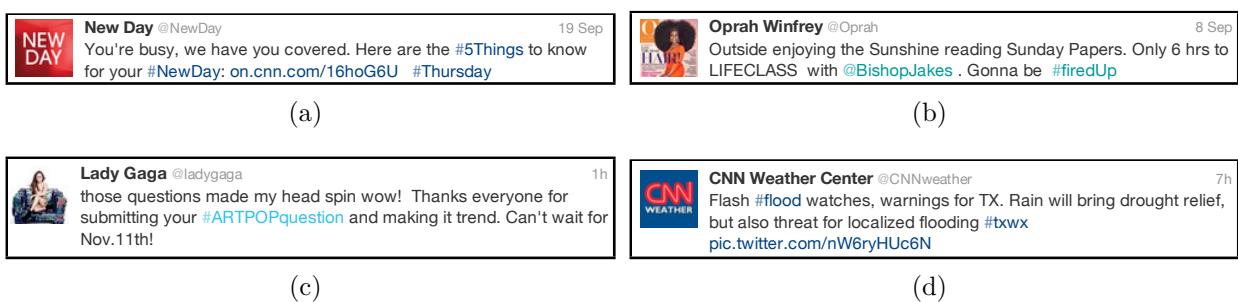


Figure 1. Example tweets on twitter.com

Some example tweets on Twitter are included in Figure 1. Note how users both add hashtags to the end of a tweet as a summary of the message and emphasize important

concepts words in the tweet by turning them into hashtags. That is, hashtags can be both intermixed within the words of the tweet and added to the end of the message.

The two memory retrieval models were also tested on tag use for StackOverflow posts. StackOverflow is a question and answer site for computer programming where users ask programming-related questions and fellow members of the community provide answers. A user with a programming-related question creates a post with their question and tags the post with a few specific programming-related tags. The tags chosen by a user on the StackOverflow site represent the primary topic of the question, such as a specific programming language, tool, software package, or framework. Some example tags for StackOverflow are *PHP*, *Arrays*, *MVC*, *C#*, and *Common Lisp*.

The screenshot shows a StackOverflow post with the following details:

- Title:** Why don't self-closing script tags work?
- Upvotes:** 414
- Downvotes:** 76
- Tags:** javascript, html, internet-explorer, xhtml
- Content:**

```
<script src="foobar.js" /> <!-- self-closing script tag -->
<script src="foobar.js"></script>
```

What is the reason browsers do not correctly recognize:  
Only this is recognized:
- Comments:** 12.8k
- Answers:** 44
- Edited:** Apr 6 at 22:00 by Darshan Computing
- Asked:** Sep 16 '08 at 6:52 by dimarzionist
- Score:** 4,680

Figure 2. Example post on stackoverflow.com

An example StackOverflow post is included in Figure 2. The site requires that for each post the user must associate at least one tag with the post, and the average number of tags per post is around 3. The author used 4 tags for this post: *javascript*, *html*, *internet-explorer*, and *xhtml*.

The StackOverflow and Twitter datasets were chosen because the human-created content between them is quite different, and I am interested in retrieval models that

generalize across tasks. However, they are also similar on several accounts: Both domains have amassed large amounts of user-created content, users generate tags for posts when creating content on the site (hashtags for Twitter and tags for StackOverflow), and the user data from the datasets are publicly available for analysis.

Also, studying user-based tag generation on these sites has relevant real-world application. Models that can accurately predict the tags that users will generate on Twitter and StackOverflow can be used as the foundation for recommendations systems on these popular sites. These systems can also help newer users by recommending proper tags for their newest content. On the StackOverflow site, for example, experts often subscribe to specific tags that interest them, and receive a daily digest of posts that tagged with those tags. Helping the user properly tag posts on StackOverflow ensures that right community sees the post, which greatly increases the chance that the question on the site will be answered quickly and correctly. For Twitter, hashtags represent streams of information that are possibly interesting to the user. Having a recommendation system that can suggest relevant hashtags to the user provides a way for the user to connect to and discover new information streams that interest them. Also, by suggesting relevant hashtags the system can help make sure a user's content is seen by other users who are interested in the same type of information.

### **Comparison Between ACT-R and Vector-Based Memory Systems.**

ACT-R's declarative memory and vector-based memory systems are substantially different in their mathematical structure. Nonetheless, both can successfully model classic behavioral patterns found in word pairing experiments (e.g., Rutledge-Taylor and West, 2008 modeling the “fan effect”).

An advantage of vector-based models is that word order can be easily incorporated into the single co-occurrence representation (Jones & Mewhort, 2007). Word order information on sites like Twitter may contain highly predictive pieces of information, as it is likely that specific words immediately precede specific hashtags. Word order has been

incorporated into ACT-R sentence-parsing models previously, but this has not been done using declarative memory. Rather, production rules were defined that ensured that the model followed a set of rules (e.g., left-to-right reading) when parsing sentences (Lewis, Vasishth, & Van Dyke, 2006). It is certainly true that at least some production rules are needed when choosing words and creating content. However, it is unexplored if some of the process of sentence construction and word selection can be represented in the ACT-R declarative memory system.

ACT-R's declarative memory system places strong constraints on how a user's prior knowledge and experience influences the likelihood that a particular memory item is retrieved (Anderson, 2007). This decay rate equation formalizes how recency and frequency of memory use relate to the prior probability that a memory will be retrieved. However, it is unexplored how a user's prior knowledge should influence retrieval for vector-based models.

**Lifetime of a Hashtag for a Specific User.** Prior research has examined the growth and decay cycle of hashtag use across users (Tsur & Rappoport, 2012). However, much less is known about hashtag lifetime for individual users. It may not necessarily follow that a hashtag's life cycle for individual users matches the life cycle across users. Further, modeling a hashtag's life within users is much more applicable to hashtag prediction, since that model can be directly incorporated into a specific user's prior likelihood of hashtag use.

**User-Customized Hashtag Prediction.** The end goal of this work is to identify a memory retrieval model that can suggest relevant hashtags to users when they wish to retrieve one. These hashtags should be customized to their specific interests and relevant to the content of the post that they are currently creating. This will undoubtedly require a combination of two primary model components: [1] a user's prior likelihood of choosing a particular hashtag, given their previous hashtag use, and [2] the likelihood that a particular hashtag is related to the context of the post being created.

A user's prior hashtag use will be an essential model component for domains such as

Twitter, where the number of possible hashtags is effectively infinite. In these domains, the model can utilize a user's hashtag history as a way to prune the infinite space of possible hashtags, and generate a much smaller and more manageable set for prediction. However, not only should the model properly take into account a user's prior hashtag use, but it should also generate new hashtag predictions when a user is most likely choosing a hashtag they have never used before. The end goal is to have a model that can properly balance the user's tag history with the contextual cues in the post in order to generate an accurate prediction.

## Roadmap

This document is separated into 7 chapters. Chapter 1 introduced the declarative memory research and motivated why it is interesting both from a theoretical and applied perspective to evaluate declarative memory models on large-scale datasets. The task to test the models was also introduced: tagging of user-generated content on StackOverflow and Twitter. Chapter 2 describes the vector-based and Bayesian declarative memory models that will be evaluated in this research, and shows how these models are related to the other common declarative memory models that are studied. Chapter 3 presents the overall method for testing the models on the StackOverflow and Twitter datasets. Chapter 4 shows the results for the first test of the models, where the model term for past user behavior is examined in isolation, and tags are predicted based purely on a user's past tagging history. Chapter 5 adds the context component to the models, and shows how model accuracy is influenced by changing various architectural components of each model. Chapter 6 describes model results when prior and context are combined and tested on both of the dataset subsets used for Chapter 4 and Chapter 5. Finally, Chapter 7 provides a general discussion of the overall findings.

## Chapter 2: Theoretical Background

The ACT-R declarative memory theory and the vector-based retrieval models will be described. The Latent Semantic Analysis (LSA) theory is closely related to vector-based retrieval models, so an overview of LSA will also be included. The theories underlying the ACT-R and vector-based models will be compared. Also, different tagging models have already been applied to various recommendation systems to tag user-generated content. Many of these models are based on the same Bayesian statistical framework that is the foundation of the ACT-R declarative memory theory. So these systems will be described and the models used will be discussed within the context of Bayesian statistical models, LSA approaches, and the ACT-R declarative memory theory.

### ACT-R Declarative Memory Theory

ACT-R (Anderson, 2007) is a cognitive architecture that formalizes how each cognitive process of the mind (e.g., memory, learning, visual and motor) interacts to produce behavior. The declarative memory system is a component of that architecture that models the timing, learning, and forgetting processes that make up declarative memory storage and retrieval. The equations that make up this system were derived from a rational analysis of declarative memory retrieval. That is, given the task of retrieving a chunk of information from declarative memory, the current context (i.e., external and internal environment state), and past experience (i.e., prior memories and exposure), what is the optimal behavior (i.e., the optimal chunk to retrieve from memory)? Using Bayesian reasoning, each chunk of information in declarative memory can be assigned a prior likelihood of needing retrieval again, given the prior history of exposure to the chunk. These chunk prior probabilities are then adjusted for the current context, so that the posterior probabilities represent the likelihood that a chunk is needed, given prior odds and adjusted by current environment state.

**ACT-R DM Model.** A formal description of the ACT-R Declarative Memory model is included in Table 1.

Table 1  
*ACT-R declarative memory model*

Common Name	Equation
Activation	$A_i = B_i + \sum_{j \in c} W_j S_{ji}$
Attentional Weight	$W_j = \frac{W}{n}$
Base Level	$B_i = \log \sum_{j=1}^n t_j^{-d}$
Constant Base Level	$B_i = \frac{p_i}{1-p_i}$
Strength of Association	$S_{ji} = \log \frac{p(i j)}{p(i \bar{j})} \approx \log \frac{p(i j)}{p(i)} = \log \frac{NN_{ji}}{N_{Row}(j)N_{Col}(i)}$
Recall Probability	$P_i = \left(1 + e^{\frac{\tau - A_i}{s}}\right)^{-1}$

The total activation ( $A_i$ ) for a chunk in declarative memory is a function of two components: base-level activation ( $B_i$ ) and strength of association ( $S_{ji}$ ). The recall probability ( $P_i$ ) that a chunk will be retrieved from memory increases with total activation ( $A_i$ ). Each term in Table 1 will be discussed in greater detail to follow.

**Base-Level Activation.** Base-level activation reflects the log prior odds of needing an observed chunk again. The default way to calculate these log prior odds is to use the standard base-level equation in Table 1. This equation formalizes how log prior odds are a function of both frequency and recency of prior exposure to a particular chunk. Chunks used more frequently (either through exposure or from a retrieval) are more likely to be needed for retrieval again. However, as time progresses and a particular chunk is no longer used, the activation for that chunk decays. In this way the standard base-level equation formalizes the time dynamics of the retrieval system, where a chunk's base-level activation evolves over time, depending on its frequency and recency of use.

For some domains it is reasonable to assume that the base-level activations of each chunk within a particular time window of interest are stable and do not change within that window. Programming language popularity over the past few years is a reasonable example

of this. Although the popularity of various programming languages has certainly changed slightly over the past few years (e.g., Clojure’s growth), it is not the case that the changes have been drastic. C#, Python, and Java are still a few of the most popular languages, Common Lisp has not gained or lost much ground in popularity over the past few years.

Further, if this is a reasonable assumption to make, it greatly simplifies the computation of the base-level activation for chunks, and can turn the computation into a tractable problem for large datasets. In these time-constant domains, one can compute the log prior odds of needing each chunk directly as the log-odds ratio of exposure to that chunk compared to exposure to all other chunks. For example, if the StackOverflow tag *PHP* has been used four times as often as the tag *Common Lisp*, then the prior odds of needing *PHP* again is  $\frac{8}{2} = 4$  times that of *Common Lisp*.

**Strength of Association.** Strength of association ( $S_{ji}$ ) reflects the amount of log-odds adjustment to the activation of a chunk, given the current context (i.e., external environment and internal state). Context for the StackOverflow domain for example is represented as each word in the title and body of a post. Context for the Twitter domain are the words in a tweet. Association strength between a chunk in memory and a single contextual element can be computed directly by calculating its context-adjusted odds ratio: the likelihood a chunk occurred with the current context ( $p(i|j)$ ) over the likelihood that the chunk occurred in any of the other contexts ( $p(i|\bar{j})$ ). For large datasets, the likelihood that a chunk occurs in any particular context reaches near-zero values ( $p(j) \Rightarrow 0$ ). So it can be assumed that the likelihood that a chunk occurs in any context but one ( $p(i|\bar{j})$ ) is equivalent to the likelihood that a chunk occurs in any context ( $p(i)$ ). This assumption was described when deriving the  $S_{ji}$  equation (Anderson & Milson, 1989), and used in recent research working with large-scale datasets (Douglass & Myers, 2010; Farahat, Pirolli, & Markova, 2004; Stanley & Byrne, 2013).

Using this assumption for large datasets ( $S_{ji} = \log \frac{p(i|j)}{p(i)}$ ), the interpretation of the context-adjusted odds ratio becomes much simpler. If this ratio for a particular tag and

context chunk is greater than one, then the log is positive, which means that this context has been observed more often with this tag than in general. If this ratio is less than one, then the log is negative, which provides a negative adjustment of total activation since this context has been observed more often in general than with this particular tag.

**Connection to Pointwise Mutual Information.** Pointwise Mutual Information (PMI) is another co-occurrence index that measures strength of association between two terms (Farahat et al., 2004). It is based on the co-occurrence count between pair-wise observations of terms, similar to ACT-R’s strength of association. The index has been used to accurately and efficiently measure strength of association information for large-scale datasets (Budiu, Royer, & Pirolli, 2007; Farahat et al., 2004). The PMI index is presented as Equation (1).

$$PMI(y, x) = \log \frac{p(y|x)}{p(y)} \quad (1)$$

Further, this PMI equation is identical to ACT-R’s strength of association ( $S_{ji}$ ) equation for large datasets. Once the number of occurrences is large enough such that the probability of observing any particular contextual element ( $p(x)$ ) is near zero, then ACT-R’s strength of association index simplifies to the PMI equation. That is, the PMI equation *is* the simplified form of ACT-R’s strength of association index ( $S_{ji}$ ).

**Attentional Weight.** The attentional weight term allows for some contextual components to be weighted more than others when the total  $S_{ji}$  activation is computed. This reflects the fact that the declarative memory system has limited attentional resources ( $W$ ) that must be spread across all of the items in context. Most ACT-R models simply weight each contextual element equally (e.g., all words in a StackOverflow post have equal weight), so each word’s attentional weight is simply normalized by the number of items in context.

**Scaling the Equations.** The majority of tag-recommendation systems use some measure of word pair associations to compute the likelihood of a tag, given current context.

As an applied concern, it is important that this measure scales well enough to handle the large volume of information present in online social media datasets. Large social media datasets such as StackOverflow and Twitter contain on the order of millions to hundreds of millions of unique word pair associations between terms and tags. If one wants to capitalize on all of the information present in the database to build the most accurate measure of relatedness between words and tags, then the association measure used must scale to this large size.

However, scale is also a theoretical concern. The declarative memory system is tasked to quickly return a single chunk of information that is stored alongside millions to billions of other unique chunks given only the current context and past history of use. To do this task, it seems plausible that a model of declarative memory must use a representation that produces accurate retrievals even when large amounts of observations are included. Further, it seems plausible that the time to compute an activation value by a computer is correlated with the time to compute an activation value by the brain. So any model of declarative memory should also be reasonably computationally efficient even with large co-occurrence datasets.

ACT-R's strength of association ( $S_{ji}$ ) and the equivalent pointwise mutual information index ( $PMI_{ji}$ ) have been shown to scale well for large corpora. Douglass and Myers (2010) implemented the large-scale version of  $S_{ji}$  in Erlang. The wall-clock time required for a declarative memory retrieval request scaled linearly with the size of declarative memory with this implementation. Retrieval times were around one second with the largest declarative memory that contained over one million associations (co-occurrence pairs).

The SNIF-ACT framework (Fu & Pirolli, 2007; Pirolli & Fu, 2003) was developed to predict user navigation of web-based hyperlinks (e.g., choosing between a set of results from a search query). The theory uses the large-scale version of  $S_{ji}$  to measure the relatedness between a user's current goals and the current context. It was shown to scale

well when working with a large-scale search query database for internet search.

As one of the largest tests for the PMI index, Farahat et al. (2004) generated the co-occurrence matrix from a 118GB local database of 10 million pages of the Stanford WebBase Project. Farahat et al. compared strength of association values using the PMI index and similarity ratings derived from human raters. Even at this large dataset size, the ACT-R declarative memory theory still scaled well enough for reasonably fast retrieval times for  $PMI_{ji}$  values.

### **Latent Semantic Analysis Memory Theory**

Latent Semantic Analysis (Landauer & Dumais, 1997) (LSA) is an alternative representation of declarative memory. Both ACT-R's Bayesian declarative memory model and LSA compute activations based on a co-occurrence matrix. However, LSA performs further processing on that matrix in order to collapse words that have similar meaning. The process starts with a set of N documents. A word by document co-occurrence matrix is built by counting the number of times each word appears in each document. The dimensionality of the full word frequency co-occurrence matrix is then reduced, while maintaining as much of the variance in the original full matrix as possible. This process, described in more detail to follow, is similar to how factor analysis takes a data table and finds the most efficient way to represent that data with as few dimensions as possible. The strength of association between two words is measured by the cosine of the two word vectors across the reduced dimension space.

Usually information is lost when the data are restructured into a smaller set of latent dimensions. However, it was shown that this process actually improves prediction and reduces noise by allowing the frequency count of highly similar concepts (words) to be pooled together and represented by a single dimension. Landauer and Dumais (1997) tested the LSA model on the Test of English as a Foreign Language (TOEFL) task of choosing the best of four possible synonyms to a target word. Model results were on par

with a large sample of applicants to American universities from non-English speaking countries (64.4% compared to 64.5%). Model performance also improved when the original dimensionality of the data was reduced, but only up to a point. Afterwards further reductions in the dimensionality of the data began to degrade performance.

**Singular Value Decomposition.** LSA uses the Singular Value Decomposition process to reduce the dimensionality of the word by document co-occurrence matrix. The technique attempts to maximize the amount of variability in the data left after forcing the original dimensionality of the data to be reduced. The best way to remove a dimension while maintaining the most variability is to look for words with highly similar co-occurrence patterns and deduce that these words represent the same concept. For example, synonyms such as *#photograph* and *#photo* would appear interchangeably across documents, so these terms would be collapsed to a single dimension and frequency counts would be pooled. By pooling co-occurrence counts the model is able to generate better predictions with less data.

**Word Order.** Although the LSA model has been shown to perform well as a way to measure similarity between two terms, one of the main criticisms is that it is a “bag of words” model. This means that the model does not account for or pay attention to word order when building the base word by document matrix or consequently, when performing the SVD dimension reduction. Other models that can account for word order have been shown to perform better than “bag of words” models on tasks that do not even appear to require grammatical structure such as word order to do well. For example, Jones and Mewhort (2007) compared a vector-based model called BEAGLE directly to LSA, training and testing both models on the same corpus used in Landauer and Dumais (1997). The context-only BEAGLE model (no word order) performed nearly equivalent to the LSA model at the TOEFL task (55.6% vs. 55.3%). However, even in a multiple-choice domain where the task is to identify synonyms (very little grammatical structure in the task), adding word order when training the model improved task performance to 57.5%. It seems

reasonable to expect that task performance will improve much more on tasks where word order is used (e.g., generating a hashtag in the middle of a sentence when composing a tweet).

**Scaling Issues.** Another issue with LSA is that the SVD matrix decomposition technique is computationally expensive. As the size of the original word by document matrix increases, the SVD computation becomes more and more time consuming. If the domains where LSA is applied contained small enough datasets where SVD computation could be computed in a reasonable amount of time, then this would not be much of a concern. However, with large human-created datasets such as Twitter, StackOverflow, and Wikipedia, the total size of the dataset far exceeds the maximum size that can even be computed by an LSA approach. Budiu et al. (2007) used the first six million pages of the Stanford WebBase corpus to compare performance between LSA and PMI on the TOEFL, and the LSA model could not be implemented with this size training dataset. Instead, they compared the measures on the TOEFL by training on the smaller Touchstone Applied Science Associates (TASA) corpus created from 60,527 samples of text from high school to early college textbooks. This is one of the available corpora on the LSA website and is intended to summarize the general reading up to the 1st year of college (Budiu et al., 2007).

When Budiu et al. (2007) compared the simpler PMI (i.e.,  $S_{ji}$ ) measure to LSA on smaller TASA-sized datasets, LSA performed much better (23% vs. 60%). This was one of the primary motivating reasons that LSA was developed in the first place, since reduced dimensionality model outperformed the full dimensionality model on TASA-sized datasets. However, once the dataset surpasses in size what is computationally feasible with LSA, the much simpler PMI measure can still be computed, and the performance for PMI becomes on par and even surpasses that of LSA. Budiu et al. (2007) also trained the PMI measure on the much larger Stanford WebBase corpus and tested the measure on the TOEFL. This PMI model increased accuracy from 23% to 51% when training the dataset on the smaller TASA compared to the larger Stanford dataset. Further, when Turney (2001) trained the

PMI measure on the even larger AltaVista index of 350 million webpages, the accuracy on the TOEFL increased to 73.8%, which is 10% higher than the TASA-trained LSA. This suggests that simpler models may be preferred to more complex (yet more accurate) models in large-scale domains, since the simpler models scale and can utilize all available information to generate predictions.

**Cost of Incremental Updating.** Another issue with LSA is that adding additional observations to a previously-computed reduced representation requires rerunning the SVD (Farahat et al., 2004). Since computing the SVD is the most computationally expensive operation with LSA, it is difficult to see how LSA could be implemented in incremental domains where information is periodically added. For Twitter and StackOverflow, it might certainly be useful to update the strength of association representation when new information arrives, especially for Twitter, where hashtag use and news content change rapidly.

## Vector-Based Memory Systems

“Holographic” memory systems (Plate, 1995) or vector-based memory systems represent a concept (i.e., a chunk) as a vector. The representation of the concept is distributed across all elements of the vector. This is somewhat similar to taking a column (i.e., a tag) on a word co-occurrence matrix and viewing the distribution of co-occurrence counts across the rows in the column as the representation of that concept. However, vector-based memory systems represent information in a much more compact way than a full word co-occurrence matrix.

For vector-based systems, each word is represented by an environment vector  $e_i$  that is nearly orthogonal to all other words’ environment vectors. The number of dimensions for these vectors is much less than the number of rows in a full word co-occurrence matrix, which is how vector-based systems represent information in a much more compact space. Paired with each environment vector ( $e_i$ ) is a memory vector ( $m_i$ ) that contains the

summed representation of all other environment vectors that have co-occurred with that  $e_i$  environment vector. These memory vectors can contain environment vectors from position-independent word co-occurrence information ( $c_i$ ) and word order information ( $o_i$ ), which can be combined into a single representation ( $m_i$ ). Over time, memory vectors accumulate a distributed representation of the most common environment vectors that co-occurred with them.

**Retrieval Process.** Retrieving the most likely chunk given context can be done in two ways: decoding and resonance (Jones & Mewhort, 2007). Decoding takes the memory vector for context and decodes it back into an environment vector. The cosine between that decoded environment vector and all other environment vectors is computed and ranked, and the chunk with the environment vector with the highest cosine is retrieved.

Resonance is the opposite retrieval process, where a memory vector is created from the context, and then that context memory vector is compared to all memory vectors. For example, assume that a user has written the word “zend” in a post, and she wants to assign a tag for the post. “zend” has co-occurred with the tag *PHP* many times previously, so the unordered memory vector for “PHP” ( $c_{PHP}$ ) contains the unordered environment vector for *zend* ( $e_{zend}$ ). To determine the most correlated memory vector with context, a memory vector is created from the current context ( $e_{zend}$ ). The cosine between that context memory vector and the memory vector for *PHP* will be high, since the *PHP* memory vector ( $c_{PHP}$ ) often contains the context environment vector ( $e_{zend}$ ). Since the cosine is high, the tag *PHP* will most likely be returned as the vector that resonates highest with the current context.

The resonance and decoding processes are inversions of each other. Usually both resonance and decoding will return similar rank orderings of most likely chunks given context. Jones and Mewhort (2007) recommends averaging the cosine values returned from each process for each chunk, and then rank ordering those averages to determine the chunk to retrieve.

**Addressing Word Order.** One of the strengths of vector-based systems is that word order can naturally be represented alongside word co-occurrence in these distributed representations. Representing word order is a case where both the encoding and decoding operations are applied. As a motivating example, say that one wants to encode the first and second words immediately preceding a hashtag in a tweet. Simply counting co-occurrences of the two immediately preceding words and hashtags does not include the order information of the words in the representation. With this representation, it would be impossible to tell the difference between 1-back words that immediately preceded the hashtag, and 2-back words that occurred just prior to these 1-back words. Order information is lost if a simply co-occurrence counting technique is used.

With vector-based systems, order information is included by creating an environment vector ( $e_{bind(i,\Phi)}$ ) that is a function of environment vectors for both the word ( $e_i$ ) and location ( $\Phi$ ). This function has the useful property that the resulting environment vector and the location vector can be inverted to return the original word's environment vector. Circular convolution and inverse circular convolution (i.e., circular correlation) are used by Plate (1995) and Jones and Mewhort (2007) to implement these operations. This allows for retrieval requests such as “playing  $\#\Phi$ ,” “just read  $\#\Phi$ ,” and “vector-based  $\Phi$  systems.”

For example, suppose that a retrieval request is made for “martin  $\Phi$ ” given that the model was trained on the TASA corpus. Within this corpus, the phrase “martin luther” often occurred. When this occurs, the word-order memory vector for martin ( $o_{martin}$ ) will be updated by convolving a 1-back location vector with luther ( $o_{martin} = o_{martin'} + \Phi * e_{luther}$ ). Also, the word-order memory vector for luther ( $o_{luther}$ ) will be updated by convolving martin with and a 1-forward location vector ( $o_{luther} = o_{luther'} + e_{martin} * \Phi$ ). The retrieval request for “martin  $\Phi$ ” can be done in two ways: through decoding or resonance.

Using decoding, the word-order memory vector for martin is deconvolved to the right ( $o_{martin} \circledast \Phi$ ) which returns the environment vector for luther ( $e_{luther}$ ). Using resonance, a memory vector for “martin  $\Phi$ ” is constructed ( $e_{martin} * \Phi$ ), which correlates highest with

the word-order memory vector for luther ( $o_{luther}$ ). Either way, luther is returned as the most likely term to match the phrase “martin Φ.”

New vectors generated by circular convolution are uncorrelated with all other environment vectors and have the same length. This allows the word order information and unordered context information for a chunk to be aggregated and represented in the same memory vector. Quite simply, the different types of memory vectors can be summed to create a single memory vector representation for each chunk (i.e.,  $m_i = c_i + o_i$ ).

**Addressing Scalability.** Vector-based representations compute chunk likelihoods by performing a correlation operation directly on the memory vectors. Each memory vector can be thought of as a column on a context word by tag co-occurrence matrix used for a Bayesian representation. Further, the number of rows for a vector-based representation are fixed and set much smaller than the number of rows in the full word by tag co-occurrence matrix. Since Bayesian representations have been shown to scale to over hundreds of millions of word by tag co-occurrences (Stanley & Byrne, 2013), the compressed vector-based representations should scale as well or better.

**BEAGLE.** Jones and Mewhort (2007) created the BEAGLE vector-based memory representation. Each memory vector ( $m_i$ ) is a summed representation of unordered and ordered word co-occurrences ( $c_i + o_i$ ). This model uses circular convolution and circular correlation as a way to encode and decode word order. BEAGLE performed similar to LSA when trained on the TASA and tested on the TOEFL (55.6% compared to 55.3%). Further, it was shown that incorporating word order information into the same representation resulted in minimal data loss (less than 1 percent of total predictive variance). This model provided an efficient algorithm for using a single representation for unordered and ordered information that showed performance similar to if not better than LSA.

**Random Permutation Model.** In order to represent word order information, a function must be used that converts an environment vector for a set of words and positions (e.g., “stack Φ”) into a new uncorrelated environment vector. The BEAGLE model uses

circular convolution and circular correlation as the encoding and decoding operations. However, that is certainly not the only operation that can generate new uncorrelated vectors from a set of memory vectors and positions. Sahlgren et al. (2008) used a simple random permutation method for this operation.

A formal description of the random permutation model in Sahlgren et al. (2008) is included in Table 2. With random permutations, each word environment vector ( $e_i$ ) is a large sparse vector of zeros with a few one and negative one values in random locations. In order to create a new environment vector for a word that preceded another word in a sentence, that word's environment vector is shifted to the left one position. This produces a new environment vector ( $e_{i-1}$ ) for the combination of the word and the position that is uncorrelated with the original word's environment vector and all other environment vectors.

Table 2  
*Random permutation model*

Common Name	Equation
Activation	$A_i = r(m_C, m_i)$
Memory Vector	$m_i = \sum_{i \in \text{allpast}} c_i + \sum_{i \in \text{allpast}} \sum_{l \in \text{locations}} o_{i,l}$
Unordered Context	$c_i = e_i$
Ordered Context	$o_{i,l} = e_{i-l}$
Context Memory Vector	$m_C = \sum_{i \in C} c_i + \sum_{i \in C} \sum_{l \in \text{locations}} o_{i,l}$
Environment Vector	$e_i = \text{rand}$

For example, suppose that the words “stack overflow” appear in a sentence, and I want to represent that “stack” preceded the word “overflow.” The “stack” environment vector ( $e_{\text{stack}}$ ) is shifted one to the left, producing an environment vector for “stack” preceding a word ( $e_{\text{stack}-1}$ ). That environment vector is then added to the memory vector for “overflow” ( $m_{\text{overflow}} = m_{\text{overflow}'} + e_{\text{stack}-1}$ ), so that “overflow” remembers that “stack” has preceded it in a sentence.

A resonance retrieval for a word following “stack” (“stack  $\Phi$ ”) is done by creating a memory vector out of all information in context. In this case, assume only a single piece of

order information is available, that “stack” precedes the word to retrieve. So the memory vector for context is the environment vector for one-back “stack” ( $m_C = e_{stack^{-1}}$ ). The memory vector for “overflow” ( $m_{overflow}$ ) will be correlated the highest with this context vector ( $m_C$ ) since “overflow” contains this vector.

**Comparison between BEAGLE and Random Permutations.** Random permutations uses one of simplest operations possible to create a new environment vector. This operation is much less costly than circular convolution used by the BEAGLE model, which allows random permutations to scale better and handle larger datasets. Further, even though random permutations is a simpler representation than BEAGLE, it also performs better when trained on the same sized corpus.

Recchia et al. (2010) found that random permutations could use smaller vectors than circular convolution approaches (e.g., BEAGLE) to produce equivalent performance (better compression), and also scale to larger datasets (better computational efficiency). The vector length by performance tradeoff was tested on a simple paired associative retrieval task. Retrieval accuracy for the random permutation vector-based models remained high relative to the circular convolution model even after reducing each environment vector’s length from 2048 to 1024. This indicates that vector-based representations using random permutations compress better than those using circular convolution.

To test computational efficiency, each model was trained on a subset of the Wikipedia corpus, and tested on the Test of English as a Foreign Language (TOEFL) assessment (among other tasks). The random permutation model was trained on 2.33 GB corpus. The BEAGLE model was unable to train on a corpus this large, so both models were also trained on an additional 35 MB subset of the Wikipedia corpus. The random permutation model performed as well or slightly better than the BEAGLE model when trained on the smaller corpus. The model’s performance also significantly improved when trained on the much larger corpus that was computationally intractable for the BEAGLE model. These results make a strong case to favor random permutations over circular convolution when

implementing vector-based memory representations.

**Connection to LSA.** Both vector-based models and LSA calculate word similarity on a reduced matrix. For LSA, the original word by document matrix is compressed to a factor by document matrix and the number of factors is much less than the number of words. For the StackOverflow and Twitter datasets, the number of documents are the number of different tags.

For random permutations, each tag is represented by a compressed vector with a length much less than the number of words. Note that if the length of a word's environment vector was the number of unique words, then each word could be represented by associating it with a particular row in the column vector. If done in this way, building a matrix of all words' environment vectors by tags would recover the original word by document matrix (Kanerva, Kristoffersson, & Holst, 2000). However, each word's environment vector is compressed using a length much less than the number of unique words and represented by using a combination of a few one and negative one values at a unique combination of positions.

Information is lost with this compression in both cases: by starting with a compressed representation with vector-based models or by reducing to a compressed representation with LSA. However, the way that the compression works in these two cases is not the same. LSA uses SVD to find words that have highly correlated tag occurrence distributions (i.e., latent concepts), and then pools those co-occurrence counts into a single dimension. A vector-based approach like BEAGLE or random permutations essentially randomizes the mapping of the words on the reduced set of dimensions, and does not take into account word similarity when generating the mapping. Nonetheless, random permutations have been shown to perform similar to if not better than LSA when trained on the TASA and tested on the TOEFL (Jones & Mewhort, 2007; Sahlgren et al., 2008).

This is a counterintuitive result. It seems plausible that an SVD technique that takes into account word similarity when reducing the matrix should generate a reduced matrix

that has more information than a randomly-compressed matrix. This does not seem to be the case in testing however (Jones & Mewhort, 2007; Sahlgren et al., 2008). Nonetheless, Kanerva et al. (2000) suggested to perform an SVD reduction technique on a compressed random permutation vector. This approach might be worth investigating, as it would reduce computational requirements even further, and possibly improve performance by pooling co-occurrence counts from latent topics together.

### **Comparison of Vector-Based Models and ACT-R**

Vector-based models can be thought of as a compressed representation of the full co-occurrence matrix used for the ACT-R declarative memory system. Each model computes an activation for each word to retrieve, given context and (for ACT-R's Bayesian model) prior knowledge. Since both models produce activations for each chunk, one can be swapped out for another and used as the declarative memory component of the ACT-R infrastructure. For example, Rutledge-Taylor and West (2007) used a variant of the BEAGLE vector-based model as the DM component of ACT-R. Rutledge-Taylor and West showed that an ACT-R theory with a vector-based memory system can successfully model the fan effect.

The fan effect describes a behavior of human memory where memory items are more difficult to retrieve if they have occurred in a larger number of different contexts.

Rutledge-Taylor and West (2008) used Anderson (1974)'s classic experimental paradigm to test the fan effect. In the task, participants are exposed to a set of person and place pairs (e.g., "the hippie is in the park"). They are later asked to recall if a particular person and place pair was presented to them before. The main manipulation is the "fan" of the person and place for each pair. The fan is defined as the number of different contexts that each item in the pair is presented in. For example, if items were presented to the participant where the "hippie" was both in the "park" and in the "store," then "hippie" would have a fan of two. Recall accuracy decrease and time to recall increase as the sum total fan of the

person and place increases.

Both the ACT-R declarative memory theory and vector-based models produce the fan effect due to the effect that fan has on the co-occurrence matrix. As a particular word (e.g., “hippie”) is seen in more and more contexts, the co-occurrence count of the word across all contexts is spread across multiple contexts. This means that when a retrieval for a context given the word occurs, multiple contexts have relatively high activation. This makes it more difficult to test if a signal (e.g., if a word appears in a specific context) is higher than the background noise (e.g., all other contexts that the word has appeared in), which makes the retrieval more uncertain.

Both vector-based and Bayesian models produce the fan effect since they both are based on co-occurrences. However, vector-based models do further processing on that matrix, reduce its dimensionality, and use a correlation instead of a log odds value to compute the activation. Nonetheless, vector-based models still produce the fan effect because at their core they are still computing activations based on co-occurrences.

**Base-Level Activation.** Vector-based models and ACT-R’s current Bayesian DM model are not identical however, and these differences need to be addressed in order to better evaluate the relative benefits of the two models. In particular, the frequency and recency of retrievals for each chunk are not currently used for vector-based models when calculating activation.

The frequency and recency of how content has been used in the past can be a strong predictor of the future importance of that information. The recent content in tweets on a news site like Twitter should be more relevant, for example. Efron and Golovchinsky (2011) tested this claim by evaluating a set of query likelihood models on returning relevant Twitter tweets from search queries. They showed that a model incorporating recency information of the tweet produced more relevant results than models that did not pay attention to this information.

ACT-R’s Bayesian model has a strong theory with respect to recency and frequency

information. It uses this information to calculate a chunk's prior activation (i.e., prior likelihood of being needed again, independent of current context). Vector-based models compute activation based entirely on context, so chunks that resonate with context will be retrieved, irrespective to the frequency and recency of use of that chunk. ACT-R's prior activation can also be computed for each user, so that hashtags that have been used frequently and recently for a particular user are rated with higher activation. Vector-based models have focused on computing the associative strength of a chunk and specific contextual elements. It is unexplored how accuracy changes for these models once the information about a user's prior behavior (e.g., ACT-R's base-level activation component) is added.

**Word Order.** One strength of a vector-based approach over ACT-R's Bayesian DM model is that word order can be naturally represented alongside unordered co-occurrence information. With the ACT-R model, for example, it is unclear how to represent that a particular word appeared just before a hashtag. To represent word order in a Bayesian model, an additional dimension could be added to the co-occurrence matrix that represents word position. However, adding word-order information to ACT-R's default Bayesian declarative memory model is not common. Instead, when word order information is used to model reading for example, production rules are defined so that words are read from left to right (e.g., Lewis et al. (2006)).

**Activation Calculation.** Vector-based and LSA retrieval models calculate chunk activation by correlating a representation of the current context with each memory item. Bayesian retrieval models like ACT-R calculate activation by computing each chunk's posterior odd adjustment likelihood, given the context. These two computations are certainly not equivalent, even though they are most likely correlated with one another.

Using a correlation can be problematic due to range effects in observed co-occurrences. If there are particular words for a tag that co-occur much more often than all of the other words with that tag, that single large value diminishes the power of all other

words to contribute to the correlation calculation. This does not happen with Bayesian retrieval models because each pair of word and tag values is normalized by the total number of words that appeared with that tag and total number of tags that appeared with that word (see the strength of association equation in Table 1). The high-frequency words are often removed from co-occurrence matrix for vector-based models to reduce this effect.

## **Recommendation Systems**

A model that predicts a user-chosen hashtag is a specific type of a recommendation system. General recommendation systems have grown in need with the growth of the web. Users on a site need quick access to specific pieces of information, and the site contains much more information than they can process or care about. Recommendation systems are used to tailor the information presented to what is most relevant to the user (Pazzani & Billsus, 2007).

**The Netflix Prize.** The Netflix Prize (Bennett & Lanning, 2007) is an example of a recommendation system where prior user behavior and current context was used to present the most relevant information to the user. Netflix is an online subscription service where users can stream television episodes and movies. Each user of the site is only interested in a small portion of the entire library of content that Netflix provides. In order to provide a high quality user experience on the site, Netflix created a content recommendation service. This tool recommends to the user movies and television episodes that most likely match their specific interests.

Netflix wanted to improve the accuracy of their recommendation system, so they held a competition where teams could try to improve on the accuracy of the service. The winning team utilized a statistical model that performed a type of factor analysis to identify important dimensions in the data (e.g., drama, action, cult classic) and place values for each user along each dimension. This method used prior user behavior to classify a user along the dimensions of the reduced space, and current user behavior (e.g., just

watched) to understand what they were interested in watching at that moment. By using a factor analysis technique and combining these indicators, the model improved on Netflix's original recommendation system by 10%.

**Recommending Followers on Twitter.** Recommendation systems based on the content of user-created posts have been successful in recommending people on Twitter for a user to follow. Hannon, Bennett, and Smyth (2010) built a Twitter follower recommendation system by creating a profile of each user based on the words they used in previous posts. The recommendation system would then suggest other Twitter users to follow that had similar profiles to a particular user. The profiles were created by generating a word frequency matrix of words by (user + followers + followees), and then providing that matrix as input into the Lucene text search engine framework.

Lucene (McCandless, Hatcher, & Gospodnetic, 2010) is a commonly-used open source software tool that measures the similarity between a query and candidate documents in a database. It uses the term frequency-inverse document frequency co-occurrence measure to quantify the similarity between a word in the query and a possible document to retrieve. This measure is the product of two terms: the frequency count of the word in the document and the log inverse of the ratio of the number of documents that the word appeared in to the total number of documents. The term frequency component measures the strength of association between a word and a document. The inverse document frequency component measures how much information is contained within a specific word, independent of document.

When Hannon et al.'s system recommended a set of individuals for a user to follow, it collected all of the words used in posts from that user, constructed a search query, and provided that query to Lucene. Lucene then returned the most likely users that share similar words in posts within their Twitter network with the words provided in the query. Using this simple technique based on a standard text search engine allowed the recommendation system to accurately predict the people that a particular user follows at

25% accuracy.

**Hashtags for Recommendation Systems.** Efron (2010) showed how hashtags might be used as a type of recommendation system for Twitter. Users constructed a query of text that represented a search for a topic that they were interested in. The system then retrieved the most likely hashtags that were associated with that search query. The model is based on building a word co-occurrence matrix of words by hashtags, and then uses that matrix to assign a similarity score for each hashtag when provided with a search query. Participants graded how relevant the hashtags returned were to each search query. The results suggest that hashtags contain useful information that help classify the gist of a post. Although this model does not predict the specific hashtags used when composing a tweet, a model like this could certainly be tailored for that purpose.

## Hashtag Prediction

Recommendation models that attempt to predict a user's chosen hashtag have recently been created and tested for a few social media sites. Google has even deployed a hashtag recommendation model to help users properly tag posts on their Google+ social media site (Google, 2013). A tag-suggestion tool has also been deployed for the tex.stackexchange.org and superuser.com StackExchange sites. Models for Twitter and StackOverflow have been proposed and tested, but none have yet been deployed. Recommendation models for StackOverflow, Google+, and Twitter will be described in more detail to follow.

**StackOverflow.** Kuo (2011) was the first to model tag use for StackOverflow posts. His set of models were originally designed for next-word prediction for large text document collections. Kuo examined the accuracy of three different commonly-used models for next-word prediction: a Bayesian model, a K-nearest-neighbors model, and a Latent Semantic Analysis (LSA) model. In order to structure the models to work for StackOverflow tag prediction, the models would request the most-likely next word after a

user created a post, and restrict the space of possible words to only tags. He tested the set of next-word prediction models to generate predictions for the most likely tags that a user would choose when asking a question on the site. Interestingly, the Bayesian co-occurrence model outperformed the more complicated and computationally expensive LSA and K-nearest-neighbors models. This Bayesian model can accurately predict a user-chosen tag at 47% accuracy.

***Stanley and Byrne (2013)'s ACT-R DM Model for StackOverflow.***

Stanley and Byrne (2013) also modeled a user's chosen tags for a post on the StackOverflow site. They modified the standard ACT-R model in Table 1 in order to more accurately predict tag use on StackOverflow. A formal description of the modified ACT-R Bayesian model is included in Table 3.

Table 3  
*Stanley and Byrne's StackOverflow tag prediction model*

Common Name	Equation
Activation	$A_i = B_i + \sum_{j \in T} W_j S_{ji} + \sum_{j \in B} W_j S_{ji} - O$
Base Level	$B_i = \log \frac{p_i}{1-p_i}$
Strength Assoc.	$S_{ji} = \log \frac{p(i j)}{p(i)} = \log \frac{N N_{ji}}{N_{Row(j)} N_{Col(i)}}$
Co-occurrence	$N_{ji} = \sum \text{observed}_{ji}$
Observations	$N = \sum \sum N_{ji}$
Attentional Weight	$W_j = W \frac{E_j}{\sum E_j}$
Scaled Entropy	$E_j = 1 - \frac{H_j}{H_{max}}$
Entropy	$H_j = - \sum_{i=1}^N p(i j) \log(p(i j))$
Offset	$O = \frac{1}{5} \sum_{i \in top5} A_i$

The activation of each tag ( $A_i$ ) is a function of four components: That tag's base-level activation ( $B_i$ ), contextual activation for the words in the title, activation for words in the body, and an offset term ( $O$ ). This model has been modified from the standard ACT-R model (Table 1) in four ways: Two separate contextual components were used (words in title and body of post), an offset term was added, an entropy measure was used to compute

the attentional weight ( $W_j$ ), and the base-level activation of a tag was based on global past tag use and not customized to a user's past tagging history.

The offset term was added to normalize the mean activation across retrievals. This allowed the use of a logistic regression statistical technique to calibrate the weights for each of the terms. The title and body contextual components were separated since it seemed plausible that the words in the title would be better predictors of the tag used than the words in the body. It turned out that the words in the body were a better predictor of tag use (Stanley & Byrne, 2013), most likely because there are many more words in the body than the title of the post.

The scaled entropy measure  $E_j$  was chosen for the attentional weight ( $W_j$ ) term because it has a strong theoretical foundation (Dumais, 1991) and is parameter free. It is based on Shannon's Information Theory, which measures the amount of information content (predictiveness) of a word. So the scaled entropy ( $E_j$ ) of a context word can be thought of as the amount of predictiveness that that word has to *any* tag. This means that a stop word like "the" will have a scaled entropy measure close to 0, while a highly-predictive word like *macro* will have a scaled entropy measure close to 1. Looking at  $W_j$ , if  $E_j$  is low then  $W_j$  is low, meaning that the scaled entropy measure helps allocate limited attentional resources to predictive contextual elements.

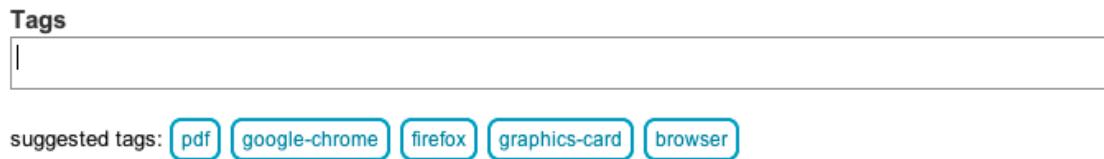
**Comparison Between Kuo's and Stanley and Byrne's Models.** Since Stanley and Byrne (2013) made several modifications to standard ACT-R model, it was more sophisticated than both the standard ACT-R model and the Bayesian model used by Kuo (2011). Also, it was unclear if Kuo used both the words in the title and the body of the post as separate components for his models, or if only the words in the title were used.

Stanley and Byrne's model also scaled much better than Kuo's Bayesian co-occurrence model. Because of this, 100 times more posts were used to build the co-occurrence matrix (1,000,000 versus 10,000), which significantly increased the model's performance. After including the modifications and scaling up the model, the model can

successfully predict a user's tag for a post at 65% accuracy.

It is interesting that in both studies the best performing model was based on a Bayesian co-occurrence framework, since that is precisely the framework from which ACT-R's declarative memory system is based. This suggests that more cognitively-plausible memory retrieval models may outperform more general machine learning approaches in this particular environment (i.e., when predicting the user-chosen tags for human-created content on social media sites).

**StackExchange.** A tag suggestion tool has been deployed for the tex.stackexchange.org (StackExchange, 2013a) and superuser.com (StackExchange, 2013b) StackExchange sites. Latex, SuperUser, and StackOverflow are all part of the broader StackExchange network of question and answer sites, and they provide the same user interface and workflow when asking questions. For the two sites where a tag suggestion tool has been deployed, once a user has created a question, suggested tags appear just below the text field where the user types the tags to associate with the question. The post author is not forced to use the suggested tags, and is free to choose tags from them or tags not represented in the suggested set. An example of the interface for a created post on the SuperUser site is included in Figure 3.



*Figure 3.* Example of tag suggestions when creating a post on the SuperUser site

The exact tag-recommendation algorithm used for the Latex and SuperUser sites is not public. After exploratory testing, StackExchange appears to be using something more sophisticated than simply looking for words in the title and body of the post that are tags. The Latex and SuperUser sites are not nearly as popular as StackOverflow currently. The sites have roughly 50,000 and 189,000 questions respectively compared to 5,000,000

questions on StackOverflow.

So it stands to reason that the algorithm is relatively complex, and there could be possible scaling and efficiency concerns with the algorithm's implementation, so it has not yet been deployed to StackOverflow. Alternatively, it is also possible that StackExchange decided to try out a tag recommendation system on a few of their smaller sites first, before deploying this type of system on their most popular StackOverflow site. Nonetheless, it is encouraging that StackExchange finds tag recommendation models beneficial to the user and has started to add them to their question and answer sites.

**Google+.** The hashtag recommendation system deployed for the Google+ social media site is probably the most ambitious model of user-created hashtags to date (Google, 2013). The user interface for this recommendation system for an example Google+ post is shown in Figure 4.

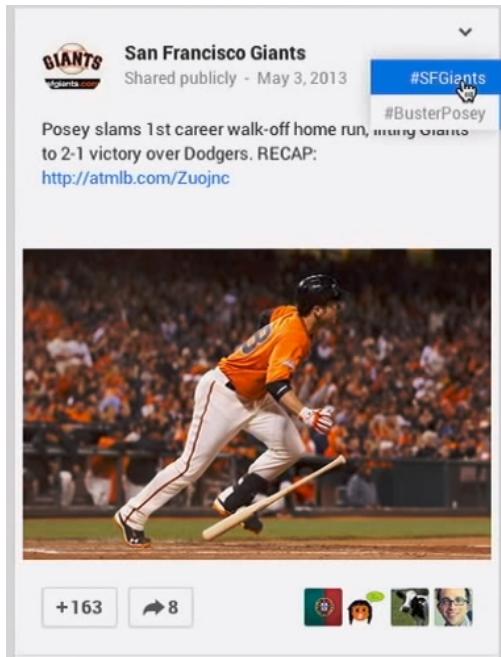


Figure 4. Example post on Google+

The workflow and user interface provided for recommending hashtags for Google+ is similar to how a hashtag recommendation system for Twitter might look. When generating content on Google+, the system automatically associates the most likely relevant hashtags

with that post. For the example in Figure 4, the system recommended *#SFGiants* and *#BusterPosey* as hashtags for the user-created post. The user is free to remove any or all of the tags that the system has associated with the post, and can also use tags that were not included in the recommended set.

Google+'s tag-recommendation algorithm is not public, but it appears to be rather sophisticated. Looking at the example post in Figure 4, *#BusterPosey* is suggested without having that specific word in the post. Also Google (2013) mentioned that the algorithm uses Google's sophisticated image recognition software to suggest hashtags for images that are included in the post (e.g., *#EiffelTower*), even when the post does not have any keywords to provide additional context.

Google is apparently highly confident in the accuracy of their system. On the Latex site, the user interface provides subdued suggestions to the user, but requires the user to explicitly choose each tag he/she wants to associate with the post. For Google+, the user interface automatically tags a user's post, and then requires the user to explicitly change those tags if he/she wants to use a different set. If tag accuracy is high enough, the Google+ technique is certainly favorable since the user does not have to spend time figuring out which tags best represent this post. However, if the model makes even just a few errors, users may become annoyed and frustrated with the recommendation system since they will have to explicitly correct it every time they generate a post.

**Twitter.** Several hashtag recommendation models for Twitter have been developed recently. One of the first models used a Bayesian co-occurrence statistical technique to predict the most likely hashtag associated with a post as a function of prior hashtag use and context (Mazzia & Juett, 2009). The model presented by Mazzia and Juett (2009) is quite similar to an ACT-R declarative retrieval model. The main potential difference is that — like Kuo (2011) and Stanley and Byrne (2013) — the global prior likelihood of a hashtag is computed without taking into account the specific user's past history. Instead, the global prior is an overall average frequency of hashtag use across all users. Further, no

recency information was used to compute the global prior, so hashtags used in the recent past were not weighted more heavily. This was most likely not much of an issue for Mazzia and Juett's study, since all hashtags were collected within a relatively short period of time (about 3 months).

Several other models have taken a tweet-centered approach to hashtag prediction, where suggested hashtags are collected from hashtags used from similar tweets. Kywe, Hoang, Lim, and Zhu (2012), Li, Wu, and Zhang (2011), Zangerle, Gassler, and Specht (2011) all store a content vector for each tweet, and then compute a word co-occurrence-based similarity score between a composed tweet and the rest of the tweets in the database. Kywe et al. (2012), Zangerle et al. (2011) used the Apache Lucene infrastructure to compute the similarity score, while Li et al. (2011) used a custom similarity score to compute similarities. Regardless of the method to compute the similarity between tweets, each method collects the hashtags used from a set of the most similar tweets, then ranks the hashtags in the set, and presents the top 5-10 hashtags to the user.

This tweet-centered approach is quite different than the hashtag-centered approach presented in Mazzia and Juett (2009). With a hashtag-centered approach, a direct association is built between the words in a tweet and hashtags that occur in tweets. With a tweet-centered approach, that association between words and hashtags is indirect: The relationship is built between a tweet and its content, and then similar tweets are assumed to use similar hashtags. One problem with the tweet-centered approach is that the storage size grows with the number of tweets, as a representation of the content in each tweet has to be maintained to later compute tweet similarity. The storage size for a hashtag-centered approach grows with the number of hashtags, but it seems reasonable that this will grow slower than the number of tweets and asymptote after collecting a large sample of tweets. So from the perspective of efficient information storage and retrieval, it seems more likely that the hashtag-centered approach used in Mazzia and Juett (2009) will scale as Twitter grows in size and use.

Kywe et al. (2012) also customized their tag prediction model to the user's past hashtag use, and was one of the first to do so when developing a hashtag-prediction model for Twitter. The set of recommended hashtags was the union of hashtags used in similar tweets and hashtags used by similar users. This was done by storing a content vector of hashtag use for each user (user-centered approach) alongside the content vectors of hashtag use for each tweet (tweet-centered approach). To generate recommended hashtags, the model would return the top-ranked hashtags from the hashtags used in 0-50 similar tweets and 0-4 similar users. Prediction accuracy improved when the model combined recommendations based on hashtags in both similar tweets and the top few similar users compared to basing recommendations on similar tweets alone.

A topics-based approach was used by Godin, Slavkovikj, De Neve, Schrauwen, and De Walle (2013), where the model predicted the most likely set of topics that were associated with a tweet, and tested if these topics matched the tweet by using human raters. The latent topics for the collection of tweets were generated using a Latent Dirichlet Allocation (LDA) statistical methodology. This is similar to a factor analysis approach in that the most likely hidden dimensions (i.e., topics) of the data are discovered by searching for reduced representations of the data that cover most of the variance. Human graders rated how well the models generated topics for a tweet related to the content in the tweet. The LDA model performed well, and generated relevant topics. When the model produced five most-likely hashtags for each tweet, 80% of the time at least one of those hashtags was suitable (as determined by human raters).

However, a topic-generating model is not performing the same task as a hashtag-prediction model. The authors argued that these general topics could be used to categorize the post, in addition to the hashtags already used by the author in the post. It is unclear though if a user would want to label their tweet with these general topics. Perhaps they would rather use more specific hashtags to be more precise in how they label the tweet and how that tweet is associated with tweets within the rest of the community. In

fact, they have already chosen to be more precise in the specific labels that they use for the tweet, since they chose to label the tweet with specific hashtags instead of general topics. Now it is certainly the case that the topics generated by an LDA model may be similar to the hashtags used in the post. However, it seems a bit indirect to build a model that predicts general topics, since embedded in the data are the exact “topics” (i.e., hashtags) that each user chose to use for each tweet. On the other hand, a topics-based approach is more general than a hashtags-based approach, since topics can be suggested for posts that do not use hashtags (which is quite common across Twitter).

**State of Research on StackOverflow and Twitter.** Stanley and Byrne (2013) used an ACT-R based Bayesian memory model to predict human tagging on the StackOverflow site, but to our knowledge no vector-based memory model has been tested on the site. Both types of models have been successfully incorporated as the declarative memory component of the ACT-R cognitive architecture (Rutledge-Taylor & West, 2007), so it should be straightforward to test a vector-based model on the StackOverflow site.

Also, a vector-based retrieval model has not yet been tested on Twitter. One of the advantages of vector-based models is that word order can be easily incorporated into the single co-occurrence representation (Jones & Mewhort, 2007). It seems quite likely that words used just before a hashtag in a tweet (combined with their position) will be highly predictive of the hashtag chosen. To our knowledge, word order has not yet been taken into account for any of the Twitter hashtag-prediction models that have already been tested.

There has been a mix of co-occurrence models created to predict hashtag use on Twitter, and the results are encouraging. However, there has been little research on how a specific user’s past hashtag use should influence the model’s prediction when that user is composing a tweet. Kywe et al. (2012) did take a user-centered view with their model, but the hashtags predicted by the user and content were analyzed separately and then the top from each group were combined for prediction. The ACT-R declarative memory retrieval theory shows how past user behavior and current context combine to produce the most

likely retrieval. Each component is summed together to generate a total activation, and then the likelihood of chunks are ranked by that summed activation. This technique is compensatory and allows for a more natural combination of the two components, where weights can be assigned to each component depending on how strongly each term predicts performance. This type of weighted additive model that combines past user behavior and current tweet context to predict the most likely hashtags has not yet been explored for Twitter.

Both StackOverflow models (Kuo, 2011; Stanley & Byrne, 2013) did not take into account the specific user's prior history when generating tag predictions. Rather their models, much like Mazzia and Juett (2009)'s Twitter model, uses the aggregate tag history across all users as the model's prior activation for each tag.

As some initial research, I have developed and tested a user-customized model for StackOverflow by modifying the model in Stanley and Byrne (2013). To incorporate the user's prior tag history into the model, the base-level activation term ( $B_i$ ) was modified from the original equation in ACT-R.  $B_i$  is based on log-odds occurrence, which in turn is based on frequency of tag use, or probability of tag use  $p_i$ . To incorporate a user's prior, a weighted average for  $p_i$  was used, where the probability of using tag  $i$  ranges from the global probability ( $pGlobal_i$ ) to the user's specific probability ( $pUser_i$ ) as the user's total tag count increases.

The global probability is the same  $p_i$  used in Stanley and Byrne (2013)'s study. The user's specific probability is computed the same way that the global probability is computed (i.e., the log odds that the tag occurred relative to the occurrence of all other tags). However, each user's probability is constrained to the set of tags that the specific user has used in the past. When a user has no past tag history, the global prior probability is used. Once a user has provided at least five tags, that user's specific prior probability is used. Between zero and five tags, both terms are used but the weight of the global prior relative to the user prior decreases as the number of tags increases.

Both linear decay and exponential decay transition functions were tried. Using either transition function, overall model accuracy improved by 5% after incorporating the user's prior tag history into the model.

This research suggests that model fit can improve when tag predictions are customized to the specific user's past tagging history. It also seems plausible that model accuracy can improve an even greater amount by using the time-dependent form of the base-level component in Table 1, in order to account for the fact that a user's tagging preferences change over time.

One of the main purposes of this research is to improve on these user-agnostic statistical models and include each specific user's past tagging history as a model component. Using customized priors that are tailored to a user's specific tagging history should be crucial for sites like Twitter where the possible tag space is infinite, but should also improve performance for more constrained sites like StackOverflow since each user is not interested in all of the possible tags on the site.

## Chapter 3: General Methodology

Each specific result of the research (to be presented in Chapters 4-6) requires a slightly different method to study. However, there is a common general methodology that applies across all results that will be described first. Different subsets of the StackOverflow and Twitter datasets were used evaluate past user behavior in isolation and to test the context component of each memory model. These datasets will be named and described in general terms. Also, the general methods used to tokenize and stem the words in all of the datasets will be described.

### **Models**

Two models were evaluated for hashtag prediction on both the Twitter and StackOverflow datasets: a vector-based model and Bayesian retrieval model. The random permutation model (Sahlgren et al., 2008) in Table 2 was used for the vector-based model, since it has shown to perform better than the original BEAGLE vector-based model (Recchia et al., 2010). The modified ACT-R Bayesian model (Stanley & Byrne, 2013) in Table 3 was used for the Bayesian model, since it has already been applied to tag use on StackOverflow posts.

The two memory models were evaluated on predicting user-chosen tags for Twitter tweets and StackOverflow posts. The co-occurrence matrices used for the context component of each model were generated from a large subset of the Twitter and StackOverflow datasets. Each model was tested by evaluating the accuracy of model-chosen tags on a fresh test subset of the datasets. Accuracy was assessed by comparing model performance with the human data.

### **Datasets**

The most recent quarterly published StackOverflow dataset (StackExchange, 2014) was used to test the models on user chosen tags for posts. This dataset contains around 5.7

million posts and 34,377 unique tags.

Unlike StackOverflow, the entire Twitter dataset is not released to the public. Due to this and the scale of Twitter, only a part of the Twitter dataset relevant to this research was collected.

**Popular Users Dataset.** A popular-users dataset was collected for both StackOverflow and Twitter. This dataset consists of all of the tweets and posts from various samples of the top users for each site. This dataset was collected by fetching all past tagging history for each of the top users. It was used to explore the growth and decay characteristics of a hashtag for individual users. The temporal dynamics of ACT-R's decay rate model for a chunk were evaluated against these data. Multiple samples were taken across a broad range of user types for each dataset to ensure that the model results were not dependent on specific attributes in a single sample.

For StackOverflow, top users were identified by their reputation score on the site (based on their upvoted questions and answers) and the total number of questions asked on the site. To create the StackOverflow popular-users dataset, a total of 12 slices of users were taken at different levels of reputation and total number of questions asked. 500 users were sampled that had a reputation higher than 500, 1000, 5000, 10,000, 50,000, and 100,000. 100 users were sampled that had asked a total number of questions higher than 50, 100, 200, 300, 400, and 500.

For Twitter, top users were those that had the largest number of followers and had authored the largest number of tweets. To create the Twitter popular-users dataset, 12 slices were also taken, this time at different levels of number of followers and total number of tweets. 100 users were sampled that had a total number of followers higher than 1000, 5000, 10,000, 100,000, 1,000,000, and 10,000,000, and a total number of tweets higher than 100, 500, 1000, 5000, 10,000, and 50,000.

**Popular Hashtags Dataset.** An additional popular-hashtags dataset was collected for Twitter. The purpose of this dataset was to test the accuracy of each model's predicted

most likely hashtag(s) for a tweet. The dataset contains approximately three million tweets where at least one of the 400 most popular hashtags are used. A quota of three million tweets was selected because previously research testing Bayesian co-occurrence models on large-scale datasets has used on the order of one million documents (Budiu et al., 2007; Douglass & Myers, 2010; Kuo, 2011; Stanley & Byrne, 2013). Tweets were collected in real time until the tweet quota was reached.

Four popular-hashtags datasets were collected at four different time periods, spanning across one month. Multiple datasets were collected to ensure that the results were not dependent on a particular sample of hashtags and would generalize across different popular-hashtag datasets and different time periods. Each dataset required around four days to collect enough tweets to reach the quota, and one collection was done each week.

**Randomly Sampled Dataset.** Since the entire dataset is available for StackOverflow, randomly-sampled posts were used to test the context models rather than creating a popular-tags dataset (analogous to Twitter). If all Twitter data were available, then the models would have been tested on a randomly-sampled dataset for Twitter as well. However, since this is not the case for Twitter, the popular-hashtags dataset was used instead of a randomly-sampled dataset for that site, while models for StackOverflow were tested on the more challenging dataset where posts were randomly sampled across all posts created on the site.

**Dataset Descriptives.** The StackOverflow datasets had a much larger document size than the Twitter datasets overall. There were on average 9 and 158 words in the title and body of a post for StackOverflow when sampled across the entire site. For Twitter, there were 13 words per tweet for the popular-users dataset and 18 for the popular-hashtags dataset.

The average number of tags per dataset was lower overall for Twitter. For Twitter, authors used on average 1.1 hashtags per post for the popular-users dataset and 2.6 for the popular-hashtags dataset. For StackOverflow, 2.8 tags were used on average across all posts

on the site.

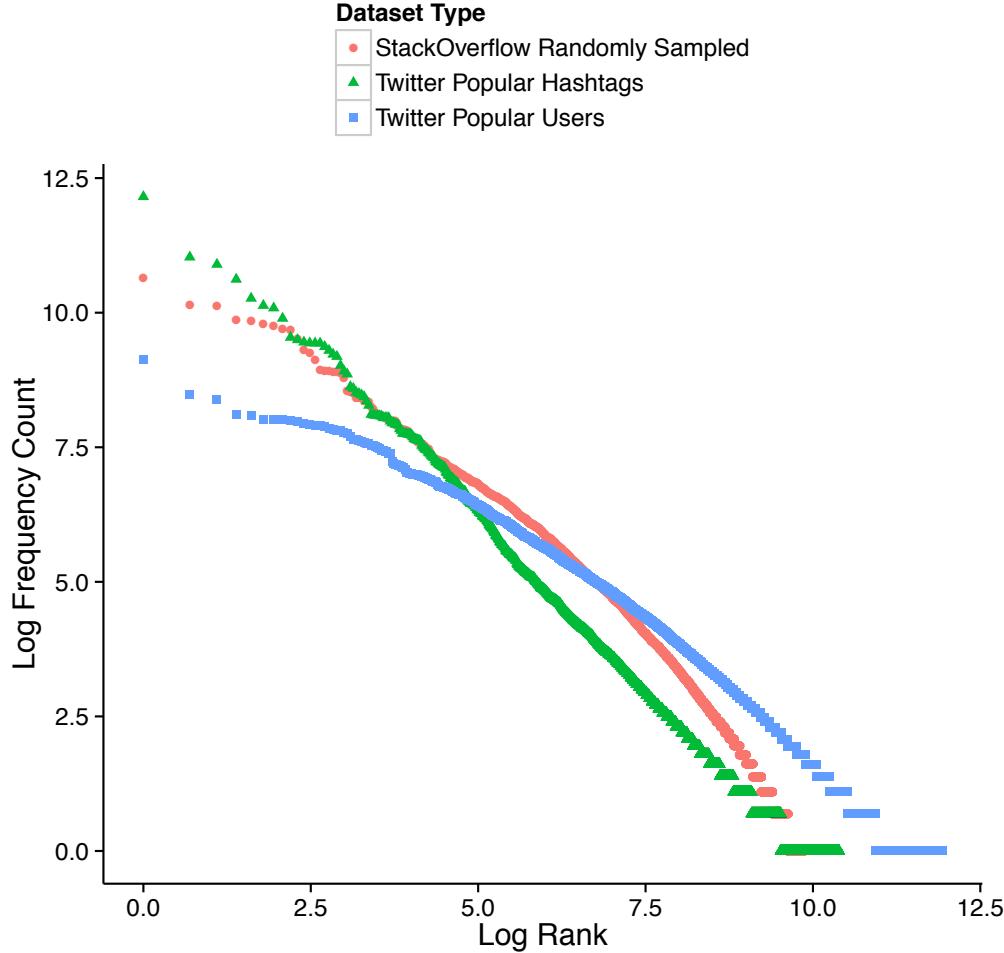
Since the document size and tags per document were higher for StackOverflow, the co-occurrence matrix for StackOverflow contained many more observations compared to Twitter. Taking 1 million posts across the site, the StackOverflow co-occurrence matrix contained 835,000,000 total observations and 55,000,000 unique combinations of context words and tags. For Twitter, when building a co-occurrence matrix for the popular-hashtags dataset from 3 million posts, the matrix contained 39,000,000 total observations and 4,000,000 unique combinations.

Hashtags for Twitter were both placed at the end of the tweet and embedded within the middle of the tweet. For the popular-hashtags dataset, 11% of hashtags were the last word in the tweet and the rest were embedded within the tweet (not at the end). Tags for StackOverflow were always placed at the end of the post, since the author chose tags by placing them in a separate text field before submitting the post.

Each of the four Twitter popular-hashtags datasets contained the 400 most-popular hashtags at that time. There was only a small amount of overlap between the hashtags chosen for each of the datasets. Only 20% of hashtags occurred two or more times across the four popular-hashtags datasets and the majority of those (66%) occurred in only two of the four datasets.

The StackOverflow dataset contains many orders of magnitude fewer unique users than Twitter. This means that the odds that the same user is sampled multiple times is less for the Twitter popular-hashtags dataset compared to all posts across the StackOverflow site. For Twitter, an author contributed an average of 2 tweets to the popular-hashtags dataset. Across the StackOverflow site, an author contributed 5 posts on average.

**Tag Frequency Distribution.** Stanley and Byrne (2013) found that the tag frequency for StackOverflow drops off sharply and approximately follows Zipf's Law. The same effect was found for the 2014 StackOverflow dataset in this research and the two Twitter dataset subsets. The results are depicted in Figure 5.



*Figure 5.* Log-log plot of tag frequency versus rank for the StackOverflow dataset and Twitter popular-users and popular-hashtags dataset subsets. Rank was assigned to each unique tag after ordering by tag frequency (most frequent given smallest rank).

Tag frequency for StackOverflow and Twitter follow a similar Zipf's Law pattern, where tag use drops off sharply after moving away from the most popular tags. However, none of the three distributions follow Zipf's Law perfectly, since none of the plots show perfectly linear lines. Nonetheless, using three separate linear models to fit the data for the three distributions captures 96% of the variance. So these log-log plots are mostly linear, and both StackOverflow and Twitter closely follow Zipf's Law for tag use frequency.

## Acquiring Datasets

**StackOverflow.** The entire StackOverflow dataset is packaged and released quarterly to the public. So the most recent packaged dataset was downloaded and the database tables were imported into PostgreSQL.

**Twitter.** Twitter provides two types of APIs for acquiring data: one based on streaming and a direct query-based interface. The Tweepy Python package provides a Python functional interface to these APIs, and this package was used to collect data from Twitter. The query-based interface was used to collect all tweets for specific popular users. At most 3000 of the most recent tweets were collected for each user, since Twitter only allows access to this number of tweets for each user.

The streaming interface was used to collect a sample of all of the tweets that included one of the top 400 hashtags at that time. Twitter's streaming API only allows at most 400 hashtags to be monitored at a time. Otherwise a larger number of hashtags would have been collected. Twitter allows the streaming interface to randomly sample about 1% of all of the real-time data that is generated on Twitter. This sampling rate was adequate to collect around three million tweets within a few days.

## Tokenization

A custom tokenization algorithm created by Owoputi et al. (2013) was used to chunk the words in the text. This algorithm was specifically created for tokenizing Twitter content. However, during testing it was also found to be robust and well suited for StackOverflow content. The initial model results with the new tokenizer were qualitatively similar to the model results found in Stanley and Byrne (2013) where the Python Natural Language Processing (NLP) toolkit (Bird, Klein, & Loper, 2009) was used. This is most likely because the language for the samples taken for both sites is informal English, and it seems reasonable to think that tokenizing all of the informal language and characters included in Twitter is a much more difficult task than tokenizing most of the words used on

StackOverflow. So the same tokenization algorithm from Owoputi et al. (2013) was used for both StackOverflow and Twitter, since performance was similar to Stanley and Byrne (2013) for StackOverflow, and it seemed beneficial to use the same algorithm for both datasets to make comparisons across datasets more equitable.

## **Stemming**

Owoputi et al. (2013)'s algorithm handles basic word stemming (e.g., removing "ed" and "ing" endings), but with the StackOverflow dataset, it is also possible to convert synonym tags to canonical base tags, since the community maintains a tag-synonym database. This can be thought of as semantic stemming, where two words with identical meaning are converted to the same canonical word. So tags used for the post were converted to their root tag when possible.

## **Handling Stop Words**

"Stop words" (e.g., "the," "a," "or") are commonly used words that co-occur with all tags (i.e., have little to no discriminating power). It is typically recommended (Bird et al., 2009) to remove stop words when building the co-occurrence matrix. Otherwise the matrix may not scale as well since these low-predictor stop words commonly appear in the text. Also for vector-based systems, memory vectors will become overly saturated with the environment vectors of the commonly-occurring stop words.

Stop words are usually removed by either filtering based on a predetermined list or filtering based on the high-frequency words that occur in the dataset. Sahlgren et al. (2008) tried both methods to remove stop words for the random permutation vector-based model, and found that the data-driven approach (i.e., building the list based on the high-frequency words that occurred in the dataset) worked much better than using a predetermined list.

Jones and Mewhort (2007) also filtered the stop words for the circular convolution model, but also mentioned that an entropy weighting technique (Dumais, 1991) has been used previously to attenuate (rather than remove) stop words. This data-driven technique

identifies stop words by calculating the discriminating power (i.e., entropy) of each word to predict any tag. Each word is then weighted (not removed) based on their discriminating power.

Stanley and Byrne (2013) used the entropy weighting technique recommended in Dumais (1991) for the StackOverflow dataset. The result maps cleanly to the ACT-R DM theory by modifying each word's attentional weight ( $W_j$ ) parameter. It also seems less arbitrary than explicitly identifying a list of stop words to remove from the analysis.

All three stop word techniques (i.e., predetermined list, high-frequency filtering, and entropy weighting) were explored for this analysis.

## Chapter 4: Past User Behavior

How does a user's prior tag use influence their likelihood of choosing that tag in the future? This question was tested by examining how well ACT-R's base-level activation component of declarative memory predicts the tags that authors choose, given each author's past tagging behavior. The model predicts that the more recently and frequently used tags have higher prior activations, and are more likely to be needed in the future.

### Base-level ACT-R model

The base-level activation component from the full ACT-R declarative memory model in Table 1 is included in Table 4.

Table 4  
*Base-level component of ACT-R declarative memory*

Common Name	Equation
Base-Level Activation	$B_i = \log \sum_{j=1}^n t_j^{-d}$
Optimized Learning	$B_i = \log \frac{n}{1-d} - d * \log L$

Both terms are methods for computing the base-level activation ( $B_i$ ) of a chunk in declarative memory. The standard form has a time dependence and accounts for the fact that the rate of presentations for a given object (e.g., the word “loop”) can change over time. The optimized learning form assumes that this rate is constant.

For both equations,  $i$  represents each chunk (e.g., the word “loop”),  $j$  is a specific observation for a chunk (e.g., the 2nd time that “loop” appears), and  $n$  is the number of times each chunk has been observed. Since the optimized learning form assumes a constant presentation rate for each chunk, it only has to keep track of the time since the first presentation of a chunk ( $L$ ) and the total number of instances observed for that chunk ( $n$ ). It does not have to keep track of the time each specific chunk instance was observed. Consequently, the optimized learning form of  $B_i$  is computationally much less expensive than the standard form, and is actually set to the default in the ACT-R Common Lisp

implementation.

The decay rate parameter ( $d$ ) represents how fast the activation for an individual presentation of a chunk decays from declarative memory. Higher values mean that chunk activation decays faster, which means that the most recently presented chunks will have the highest activation, and the total number of times that a chunk has been presented matters less. Lower decay rate values flip the importance of frequency and recency: Lower values mean that chunks that have occurred most frequently will have the highest activation, and activation depends less on the specific chunks that have been observed in the recent past.

The base-level ( $B_i$ ) component in Table 4 is different than the  $B_i$  used in Stanley and Byrne (2013) (Table 3). Stanley and Byrne assumed no time dependence, no decay rate, and that the  $B_i$  is simply a log-odds computation of the probability that each chunk is likely to be presented again, given past frequency. If a decay rate of 0 is used on the optimized learning form of  $B_i$ , then the equation collapses to  $\log(n_i)$ , which is a nonlinear transformation of the  $B_i$  used by Stanley and Byrne ( $\log \frac{p_i}{1-p_i}$ ). Both forms maintain the same rank ordering, so they should produce similar results. So the  $B_i$  component used by Stanley and Byrne can be roughly thought of as the standard form of  $B_i$  when the decay rate is 0 (i.e., a pure frequency model).

## Method

The StackOverflow and Twitter popular-users datasets were used to characterize tag growth and decay for a user on each site. The popular-users dataset contains 12 slices of popular users for both StackOverflow and Twitter. Each user's tag behavior from the popular-users dataset for each site was analyzed.

22 different values of the decay rate parameter were explored for each slice (0, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.8, 2, 5, 10, 20). The values were chosen to look at how performance changes from a pure frequency model ( $d = 0$ ), to a blend of frequency and recency ( $d \approx .5$ ), to a pure recency model ( $d = 20$ ). The decay rate

value was varied for both the standard form and optimized learning form of  $B_i$ . Only values equal to or below 1 were examined for the optimized learning form, since values above 1 for this form cause a bifurcation and are not psychologically interesting. The bifurcation occurs because at a decay rate of 1 the denominator of the first term to compute  $B_i$  goes to zero (see Table 4).

Both the standard form and optimized learning form of  $B_i$  were used to predict each user's specific tag history. To do this, a user's entire tag history was collected, and then each tag instance from oldest to newest was examined. For each of these tagging instances, the model generated an activation value for each possible tag based on that tag's frequency and recency of use. This means, for example, that the model always missed characterizing the first observed tag for each user, since at that point the model does not have any history of tag use for that user. This also means that the tag space for each user is constrained and limited to the tags that each user has used in the past.

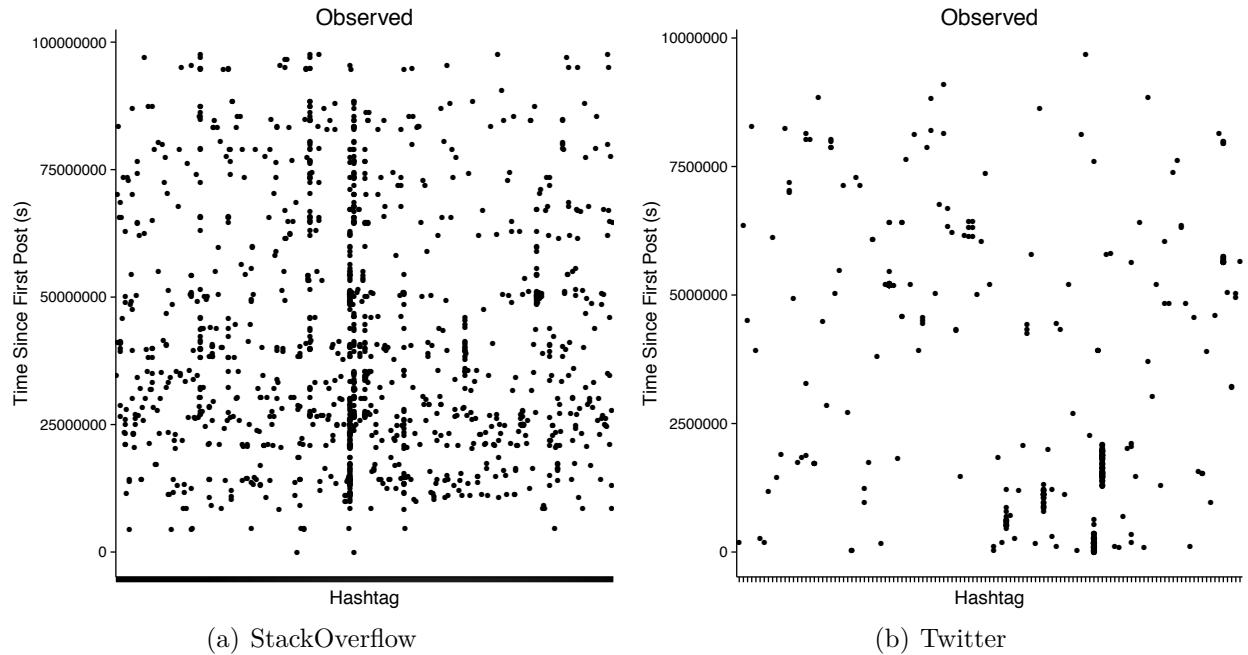
At each tagging instance, the model has activation values for a set of tags. Model accuracy was assessed by rank ordering these activations and choosing the model's highest  $N$  tags, where  $N$  is the number of tags used for that specific post. The model's chosen  $N$  tags were compared to the user's chosen  $N$  tags, and if a model's chosen tag was in the set of user's chosen tags, then the prediction was marked as correct. Otherwise, it was marked as incorrect.

This process was performed across all users in each of the 12 dataset slices for both the Twitter and StackOverflow popular-users datasets. It was repeated for each of the decay rate values and the two different forms of  $B_i$ .

## Results

**Visualizing Tag Use.** To better illustrate the dynamics of each user's tagging behavior, profile plots for a single user for both StackOverflow and Twitter are included in Figure 6. The specific value of the hashtag used for both plots is a nominal variable and not

important for this analysis. What is interesting is the large tag space for these users. There are bands of commonly-used tags for each user, but there are also many instances where new tags are used, and a large percentage of tags used are not in the most common bands.



*Figure 6.* Tagging profile for a single user. The x-axis represents the specific hashtag a user chooses for a post and time progresses vertically. A point in the plot represents when a user chooses a specific hashtag for a specific post. A user's chosen hashtag for their oldest post that contains a hashtag is the lowest point on the plot.

For example, the StackOverflow user in Figure 6 has a pronounced band where the same hashtag is chosen repeatedly over time. However there are many instances of other tags being used in parallel to the tag that is producing the band (that is repeatedly being used). The Twitter user has a few clusters where the same hashtag is chosen repeatedly but only for a brief period of time. As time progresses, a band of choosing another hashtag is observed, and then the user switches to a new hashtag after that. More recent posts for the Twitter user are more varied, and there is no longer a single hashtag that the user chooses repeatedly. This helps illustrate how challenging modeling this task can be. The model's goal is to predict at each instance what tag will be chosen, but to do so the model must predict the correct few tags from a much larger set of tags whose frequency of occurrence is

changing over time.

Although the tag space for the Twitter user is smaller than the StackOverflow user in these plots, this is not a consistent trend. These plots were chosen from the set to also show that the amount of tag variation for individual users also varies: some use many tags and some use just a few. To accurately model human performance in this domain, the model must be able to make correct predictions across all variations of user tagging behavior.

**Visualizing Model Performance.** As a first method for understanding model performance, profile plots of model's predicted tags for individual users were overlaid on that user's actual tag use. These plots for StackOverflow and Twitter are included in Figures 7 and 8 respectively. These are the same plots shown in Figure 6 with model predictions overlaid in red.

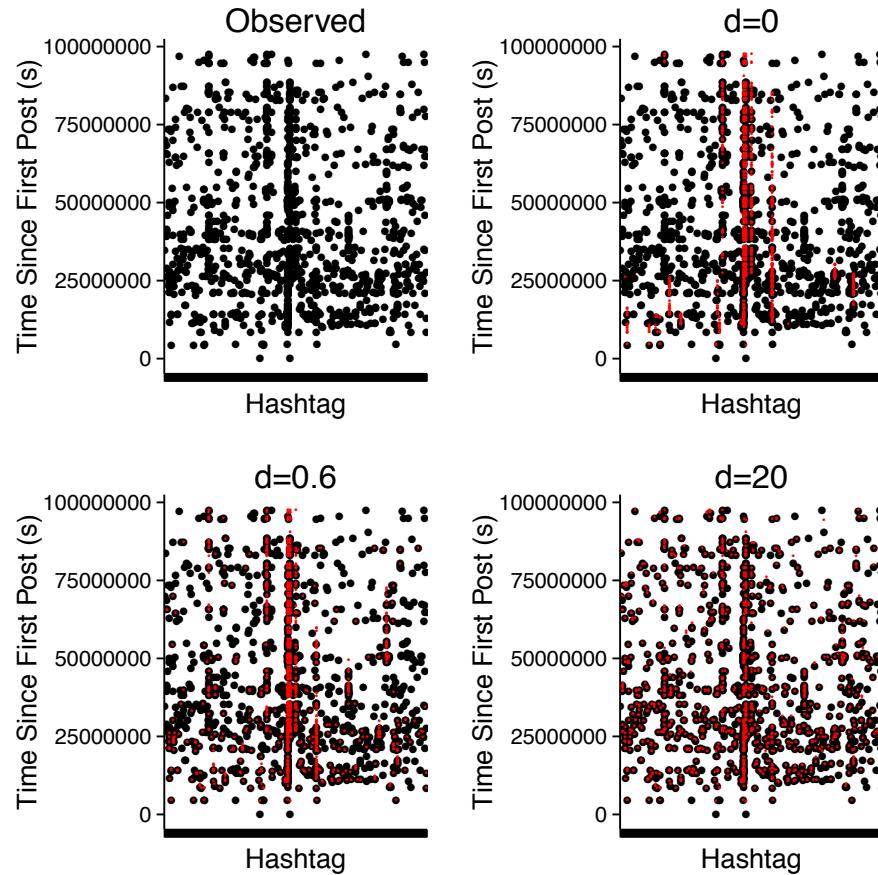


Figure 7. Model tagging profile of a single user on StackOverflow

Also shown in these figures is how model predictions change as the decay rate

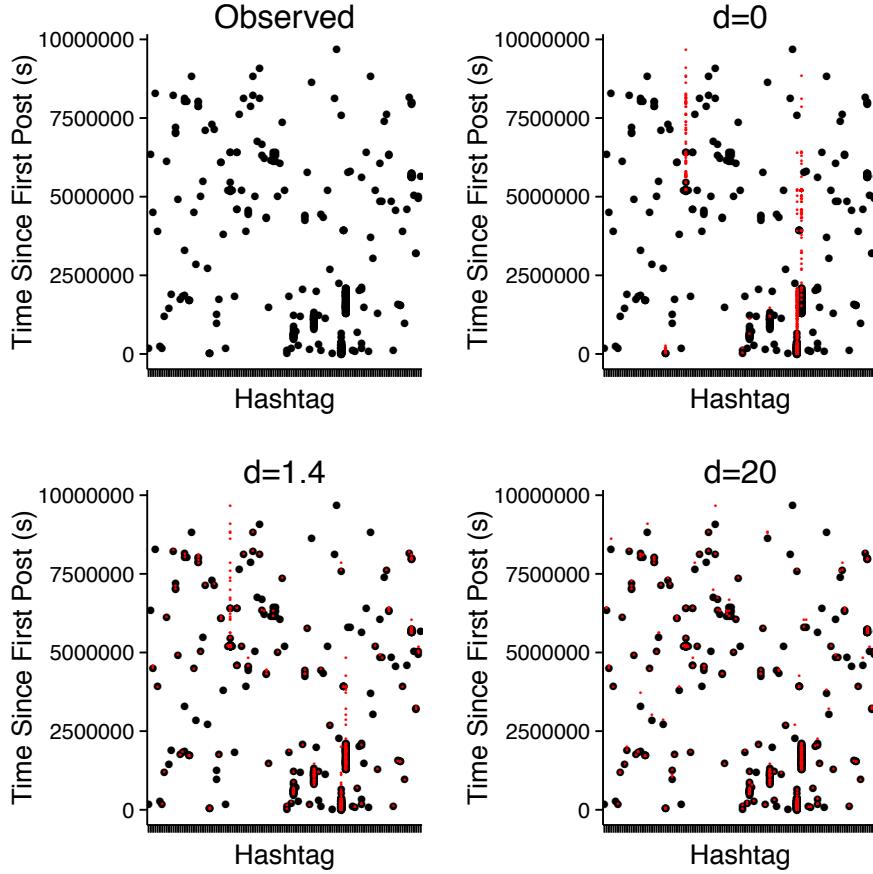
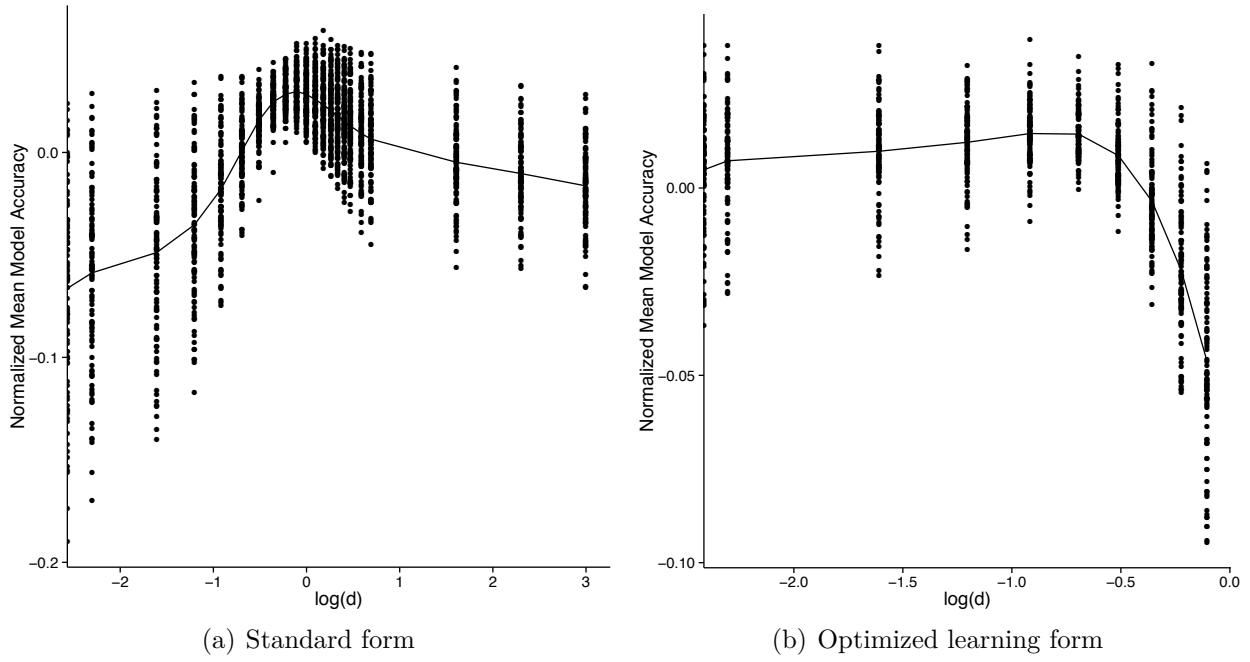


Figure 8. Model tagging profile of a single user on Twitter

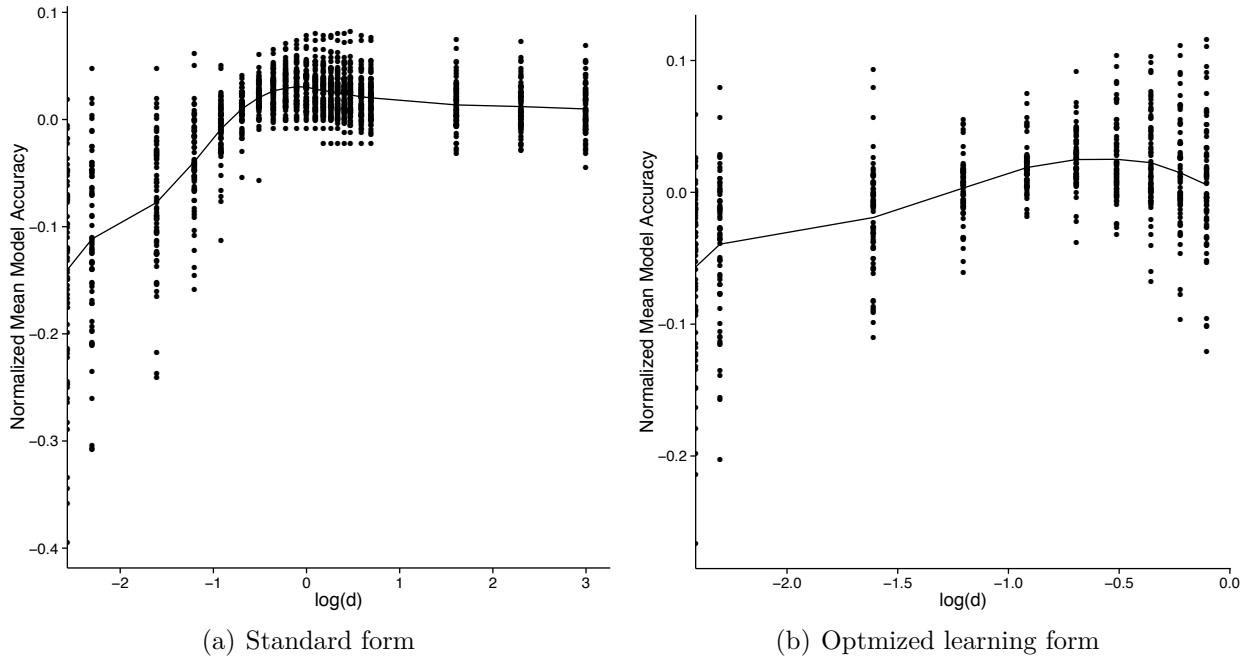
parameter changes. With lower decay rate values, the model is less likely to switch from tag to tag for each post. This makes sense because lower decay rate values lead to the model weighting frequency more than recency in the activation calculation. These plots also suggest that model accuracy is best when a blend of frequency and recency is used (i.e., a decay rate between 0 and 20). This will be more formally tested when model accuracy across decay rate values is analyzed across all users in each dataset slice and all dataset slices.

**Model Performance Across Decay Rate Values.** Moving up one level of analysis, model accuracy for all users in a single dataset slice was analyzed. A plot of the results for one of the 12 slices, for both Twitter and StackOverflow, and for both forms of  $B_i$  are included in Figures 9 and 10 respectively.

Model accuracies at each decay rate value for each user in the dataset slice are



*Figure 9.* Model performance for a single dataset slice for StackOverflow. This includes model accuracy as a function of decay rate value for each user in the dataset slice. Model accuracy is shown both when the standard form of base-level activation ( $B_i$ ) and the optimized learning form of the equation are used.



*Figure 10.* Model performance for a single dataset slice for Twitter. This includes model accuracy as a function of decay rate value for each user in the dataset slice.

included in the plots. Each user's mean accuracy was subtracted off in order to compute a normalized mean for that user. This was done so that the relative accuracy change for different decay rate values could be more easily visualized. The plotting technique is analogous to what is done when testing the main effect in a repeated measures design (i.e., each subject's mean score is subtracted off).

One large effect in all of the plots is that there is an optimal decay rate value between 0 and 20 that produces the highest model accuracy. For the standard  $B_i$  equation, model accuracy for a pure frequency model ( $d = 0$ ) is also worse than using a pure recency model ( $d = 20$ ), particularly for Twitter. It makes sense that a pure recency model for Twitter is relatively more accurate than for StackOverflow, as the hashtag lifetime for Twitter is much less than a tag for a programming language on StackOverflow.

When comparing the standard and optimized learning forms of the equation, it is apparent that a pure recency model does not work well for optimized learning. The best-fit decay rate value for the optimized learning form is also not as clear and pronounced as it is for the standard form (less of a peak). And the relative accuracy gained from using a pure frequency based model to a blend of frequency and recency is less for the optimized learning form than the standard form of  $B_i$ .

**Best-fit Decay Rate Values.** To identify the best decay rate value for the dataset slices, each user's best decay rate value was taken and the results are plotted in histogram form in Figures 11 and 12 respectively. The best-fit values are close to the 0.5 default value set in ACT-R when the optimized learning form is used. Comparing across optimized learning and the standard form, the best-fit values for the standard form are slightly higher for both the StackOverflow and Twitter dataset slices. Comparing best-fit values will be more thoroughly tested when looking at the values across all dataset slices.

**Aggregate Model Performance.** Aggregate best-fit decay rates were computed by taking the median of all user's best-fit decay rate values across all dataset slices. The median was used since there were users in the popular-users dataset that did not have

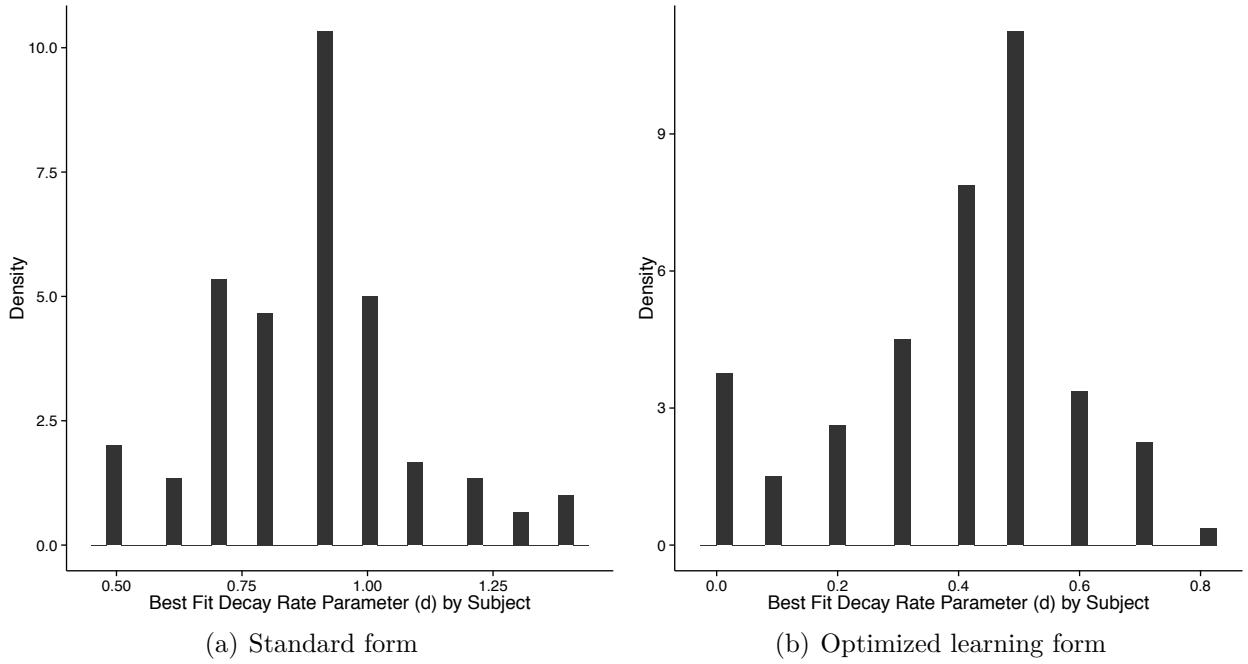


Figure 11. Best-fit decay rate for a single dataset slice for StackOverflow. The histogram represents the count of users for each best-fit decay rate value. Each bar represents a discrete decay rate value that was explored.

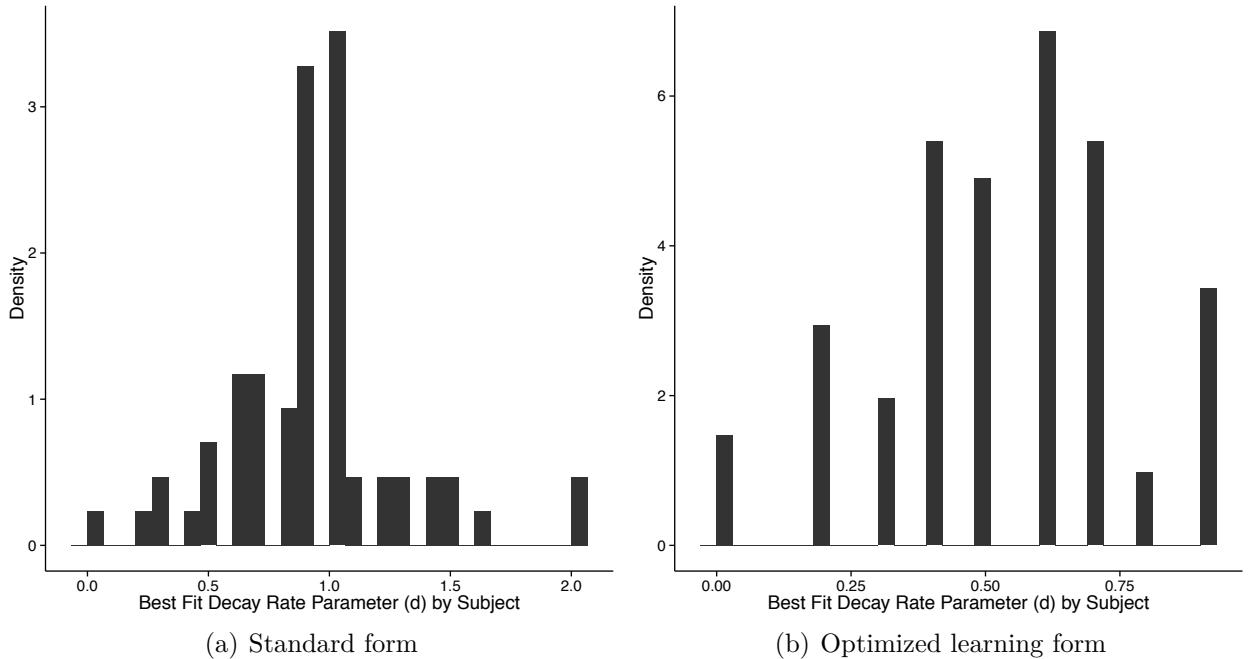
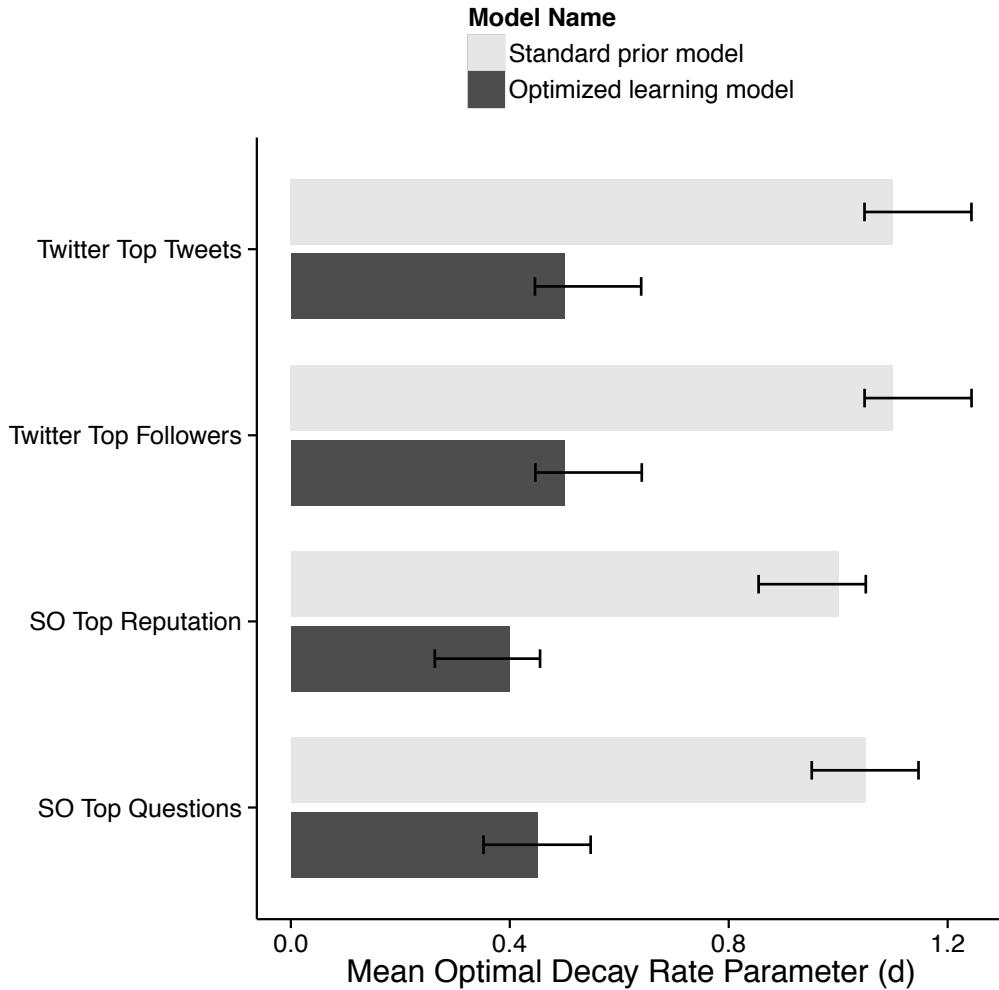


Figure 12. Best-fit decay rate for a single dataset slice for Twitter. Best-fit values were slightly higher for Twitter, so the range of the histogram is higher for Twitter than StackOverflow.

enough observations to generate stable predictions, and best-fit decay rate values for these users could be as high as 20. Using the median effectively trims these unstable values from the sample, without having to define a cut point for an outlier removal process.

Aggregate best-fit decay rate values for each model are included in Figure 13. Decay rate values that produced the most accurate model performance are lower for the optimized learning form (0.43) of  $B_i$  compared to the standard form (0.80). The optimal values for optimized learning are near 0.5, which lines up with default value used in ACT-R. So the decay rate values for the standard form are slightly higher than the default values used in ACT-R, but the values found when optimized learning is used line up nicely with the ACT-R defaults.



*Figure 13.* Overall best-fit decay rate for StackOverflow and Twitter. Results are shown for the two Twitter and StackOverflow (SO) popular-users datasets. Error bars represent the 95% bootstrapped confidence interval for the median.

Finally, model accuracy was analyzed across all dataset slices for StackOverflow and Twitter. This was done by examining model accuracy at a specific decay rate value for each model. Accuracy was assessed at each model's previously-determined best-fit values (0.43 and 0.80 for the optimized learning and standard form) and the standard form's default value (0.5). Since decay rate values were explored at 0.1 increments between the 0 and 1 range, the decay rate values used to examine model accuracy were rounded to the nearest decay rate value that was included in the run. So a best-fit value of 0.4 was used for the optimized learning model instead of the 0.43 average found from the dataset slices. Accuracy for both sites is included in Figure 14. Aggregate accuracy was computed by

taking the mean of every user's model accuracy across all users in all dataset slices.

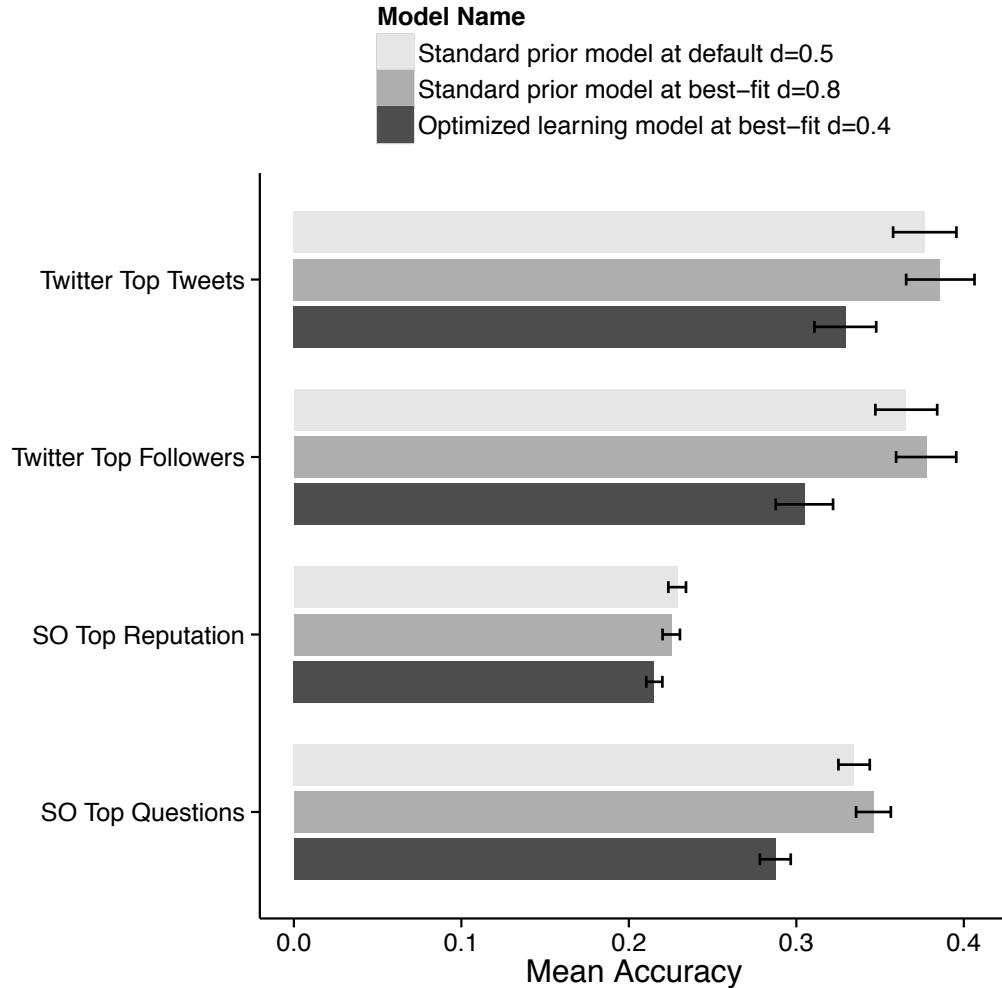


Figure 14. Overall model accuracy for StackOverflow and Twitter at optimal decay rate values. All error bars are the 95% bootstrapped confidence interval for the mean.

Accuracy for the standard form (.33) is higher than the optimized learning form (.28). This makes sense given that the standard form does not assume equal presentation rate for each chunk, and hashtags on Twitter in particular show trends of high rate of use and then shift to low rate of use. The standard form of  $B_i$  is able to more accurately account for these trends.

ACT-R sets the default decay rate value to 0.5, and the best-fit value for the standard form is higher (0.8) for these datasets. The figure shows that although model accuracy does slightly improve for the standard form when increasing the decay rate

parameter from 0.5 to the optimal value, that percentage improvement is not as large as the improvement gained from using the standard form compared to the optimized learning form. Nonetheless, model accuracy is highest when both the standard form is used and the decay rate value is increased.

## Discussion

Given the large tag space used by post authors on StackOverflow and Twitter, it was surprising that model performance is respectable even when only past tag history and no context is used. This speaks to the importance of taking into account overall past tagging history, and ensuring that the prior component of activation for each tag is customized to each user's specific past tagging history.

Also apparent for these datasets is that the optimized learning form of  $B_i$  is not as accurate as the standard form. Further, for the optimized learning form there is not much difference between using a pure frequency-based model and a model where frequency and recency are optimally blended. This makes sense given that the optimized learning equation in Table 4 collapses to a pure frequency model as the amount of time since the presentation of the first recorded chunk instance increases, and also since the timespan of recorded tag use for StackOverflow and Twitter is large (years).

So it may be worthwhile to use the standard form instead of the optimized learning form when declarative memory is modeled across a broader range of tasks. However, this should not be taken as a strong recommendation, since the performance benefits seen in this study could simply be due to the fact that the declarative memory retrievals span across a large period of time, which is not the case for all tasks. Nonetheless, the only real reason not to use the standard form is due to computational resource limitations. Since this is becoming less of an issue as hardware speed increases, it may be time to reconsider the default form of base-level learning used in ACT-R. It may be the case that the optimized learning form of the equation is no longer computationally necessary. And if the standard

form is used as the default (or simply tried in another experiment), the results from this study suggest that the optimal decay rate value should be set slightly higher (0.8) than the default when optimized learning is used (0.5).

The decay rate parameter in  $B_i$  determines how fast a presentation of a chunk in the past decays over time. Higher values of  $d$  mean that activation decays more quickly, which means that the chunks used more recently are the most active and the total frequency count of a chunk matters less. Looking back at Figure 13, the optimal decay rate parameter when the standard form of  $B_i$  is used is higher for Twitter than StackOverflow. It makes sense that the lifetime of a specific hashtag on Twitter is shorter overall than the lifetime of a tag (e.g.,  $C\#$ ) on StackOverflow, since hashtag topics are generally more transient on Twitter than StackOverflow (e.g., hashtags for specific political events or upcoming conferences). This means that the decay rate for Twitter should be higher than StackOverflow, which is what is observed.

## Chapter 5: Combining Predictors

The declarative memory models analyzed for this research consist of two main components: past behavior and context. Past tag use was isolated in the first study to see how well ACT-R's prior component of declarative memory (i.e., base-level learning) fits each user's tag use over time. For the rest of the studies the past behavior and context declarative memory components were combined as well as looked at in isolation.

Chapter 5 will examine how model accuracy is influenced by two methodological concerns (adding an offset term and examining the effect of dataset subset for Twitter) and five architectural concerns (method of handling stop words, influence of dataset size, amount of compression used for the vector-based models, method to combine the prior and context for the vector-based models, and word order). The two methodological concerns were examined to ensure that the technique to calibrate model parameters was accurate and also that results for Twitter were not specific to the subset of data collected.

The five architectural concerns were examined for the following reasons: During exploratory evaluation, model accuracy seemed to be highly sensitive to the method of dealing with the commonly-used stop words, so several different methods were examined in more detail.

Due to research from Budiu et al. (2007) and Recchia et al. (2010), model accuracy for the Bayesian and vector-based models is influenced by the number of documents in the dataset, so the effect of dataset size on accuracy was examined in more detail. Specifically, since both the Bayesian and random permutation models are being evaluated, analyzing the effect of dataset size on accuracy in this research will allow direct comparisons between the two models on a common dataset. This has not yet been possible since the two models have never been compared on the same dataset within the same study.

Sahlgren et al. (2008) showed how different levels of compression in the random permutation model influence accuracy, so this was examined for three reasons: to replicate the findings, as an additional check to ensure that the random permutation model was

implemented correctly, and also to see if the same levels of compression are valid for the much larger datasets used in this research.

One of the primary goals of this research is to identify the best method to combine the model term for past user behavior with the context term for the random permutation model, so two different ways to add the model terms were tested.

Finally, previous research has shown that including word order in the vector-based models improves performance (Jones & Mewhort, 2007; Sahlgren et al., 2008), so word order was also examined to see if the effect could be replicated and also to measure the overall importance of word order on model accuracy relative to other architectural manipulations.

## Overall Method

**Logistic Regression.** In order to best combine these predictors, a logistic regression statistical technique similar to Stanley and Byrne (2013) was used to determine the most optimal weights for each term. Stanley and Byrne (2013) weighted each term in the total activation equation (Table 3) to maximize model accuracy. Stanley and Byrne found that the optimal weights for the StackOverflow model for the prior term, title context term, and body context term were 0.62, 0.93, and 1.75 respectively.

The optimal weights for each model component were computed in the same manner as Stanley and Byrne (2013): When a user creates a tag for a post, a retrieval request for the model is made. For each request, the model returns a set of tags (all tags that have been observed for that user in the past) and activations associated with each tag. An activation value is computed for each model component (e.g., prior tag use, context) for each tag. These values are recorded, and tag instances that match what the author actually tagged are marked as a 1, while all others are marked with a 0. This process is repeated across all posts where a user chooses a tag, and across all users in the dataset. Once all of the results are aggregated, a logistic regression is run where the optimal weights for each

model component are found that maximize the model’s ability to correctly label tags as 0 or 1 based on total activation.

Model accuracy was evaluated by rank ordering all recorded tags by model activation using the optimal weights, asking the model to tag the top N posts with a 1 (where N is the total number of recorded tagging instances in the dataset sample), and then comparing the model’s chosen tags with the used tags for each post (i.e., looking at the “hits”). This is equivalent to throttling the threshold in the logistic regression such that the total number of observations labeled as a 1 match the total number of recorded instances of tagging in the dataset.

For this form of model accuracy, the model labels N posts with a 1 where the labeling is not constrained in any way within each post. That is, the labeling is relaxed across all posts, and there are no constraints where the model must label a specific number of 1s for each post based on the number of tags that the author used for that post. So model accuracy values found with the relaxed method are actually slightly conservative, since the model may be labeling more 1s for a post than the number of tags the author used for that post.

However, this relaxed metric was used for the rest of the experiments that involve combining terms because the optimal weights found by the regression are based on using the relaxed metric. If these weights are then used to compute the constrained form of model accuracy, biases may be introduced since the regression was not optimized to the constrained form, and weights may change as a consequence. Also, there is no current logistic regression technique in R that can be constrained to label a specific number of 1s for each observation where a small set of 1s occurred. To optimize parameters based on the constrained form, a full-enumeration numerical approach would have to be developed that systematically searches the space of coefficient weights and uses a custom objective fitting function to find the best fit. However, this was not pursued for three reasons: Around 2500 total regressions were run on these datasets, a custom implementation in R would not

likely run in compiled C, and even if it was implemented in a compiled language it would be computationally intractable without a closed-form solution.

**Adding Offset Term.** This logistic regression method is a computationally efficient way to find the optimal weights for the predictors. However, when the regression can not be constrained to label a specific number of 1s for each observation (as is the case here), to be used properly an offset term must be added as an additional model predictor.

The offset term keeps the activation for the small number of top tags for each post equal across posts. This ensures that the logistic regression only labels a few 1s for each post. Stanley and Byrne (2013) did this by subtracting off from each post the mean total activation for the top 5 tags for that post. However, this method is complicated since computing the total activation for a tag requires already knowing the weights for each model component that are used to compute that activation. Stanley and Byrne used an iterative approach to determine the weights used for the offset term, and kept iterating until those weights matched the weights returned after running the logistic regression.

A simplified version of Stanley and Byrne (2013)'s method was used for the results for this research. Instead of subtracting off the mean total activation for the top 5 tags for each post, the process is done in isolation for each model component. So for the prior component, the activation value used in the logistic regression is the prior activation with the mean of the top 5 prior activations subtracted from this value. The same process is used for the context component. Decoupling the offset term and computing an offset for each model component in isolation means that an iterative approach is no longer necessary, since the optimal weights of the terms are no longer needed when computing the offset.

**Datasets.** Model accuracy was tested on the popular-hashtags dataset for Twitter and a random sample of posts across the entire StackOverflow dataset. For each of the four Twitter popular-hashtags datasets, model retrievals were done across 15 runs and 500 posts within each run. For the StackOverflow randomly-sampled dataset, 5 runs of 100 posts were used, since the results for StackOverflow are more stable and retrievals take more

wall-clock time for this dataset due to the size of the co-occurrence matrix. All runs contained different posts, and all posts used for runs were not used to generate the co-occurrence matrix for the two retrieval models.

**Computing Priors.** For each randomly-sampled post for StackOverflow, the prior activation is based on that user’s prior tagging history, and not on any global prior of specific tag frequency across users. This was done so that the prior component used for StackOverflow for both the popular-users and randomly-sampled datasets was the same.

However, a custom user prior for the Twitter popular-hashtags dataset cannot be used, since the entire Twitter dataset cannot be downloaded. The popular-hashtags dataset contains samples of a large set of various users that happened to use one of a specific set of hashtags being monitored. So in order to generate a custom user prior for each sample in the dataset, that user’s entire tagging history must be accessed through the API. Since access rates are limited across that interface, it is infeasible to do this process for every user that appears in the Twitter popular-hashtags dataset. So instead, the global prior tag history for all users in the Twitter popular-hashtags dataset was used to compute prior activation values for each tag.

**Co-occurrence Matrices.** The context component for both the random permutation and Bayesian model is based on the same co-occurrence matrix of words and observed tags. Different size matrices were analyzed, ranging from 1000, 10,000, 100,000, 1,000,000, and 3,000,000 posts. The matrix for 3,000,000 posts was only used for Twitter, since the number of words in a StackOverflow post is much higher than Twitter on average, and the space and computational time required when using a co-occurrence matrix of 3,000,000 posts for StackOverflow is too large for a current high-performance desktop, even with highly optimized representations. Already 835,500,000 total co-occurring word tag observations were used to build a co-occurrence matrix of 1,000,000 StackOverflow posts. It was also the case that 1,000,000 posts for StackOverflow was already around 1/7 of the total posts created on the site, so it is reasonable to assume that the results would not

change greatly for StackOverflow if 3/7 of the total posts created were used to build the co-occurrence matrix instead, especially since 835,500,000 co-occurrences were already used with 1,000,000 posts.

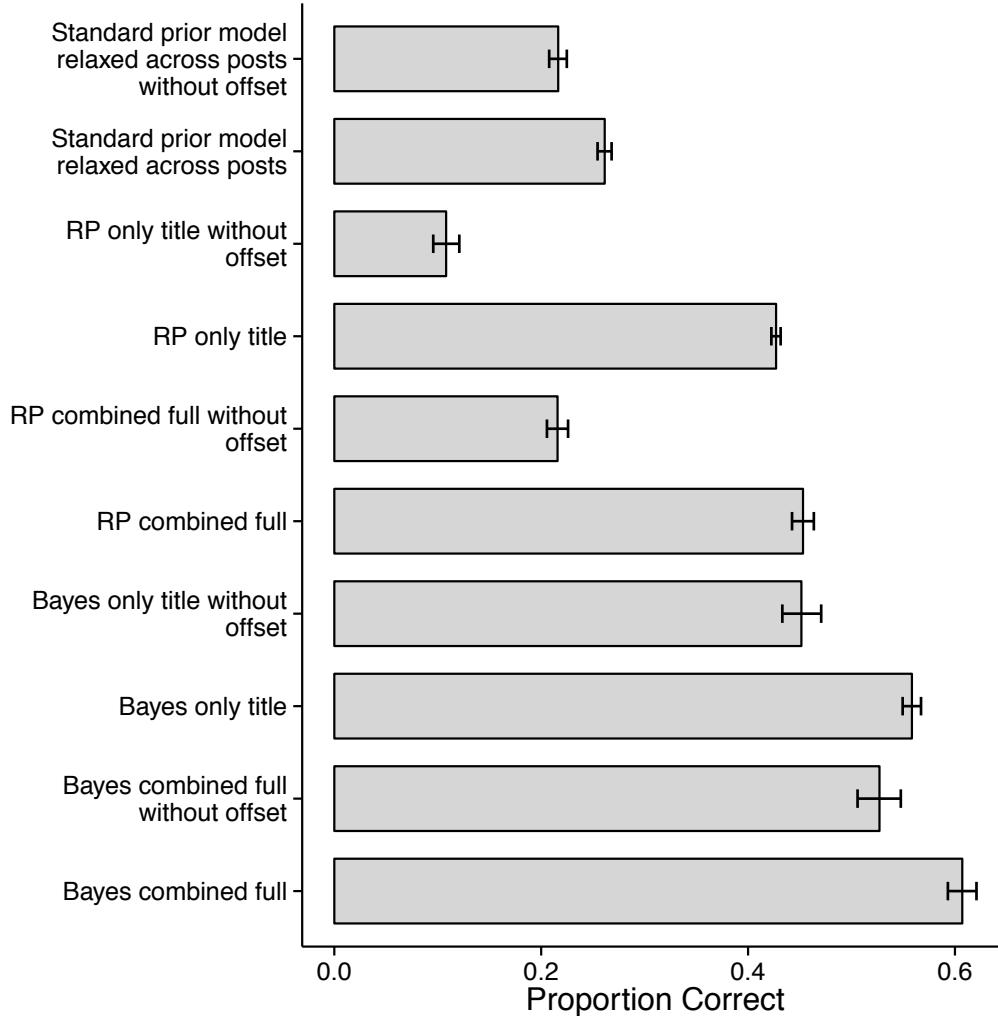
## **Adding Offset**

The first methodological concern was the offset component. To test the effect of adding the component on model accuracy, the offset component was added to both models, and model accuracy was examined before and after the addition.

**Method.** The four Twitter popular-hashtags datasets and the StackOverflow randomly-sampled dataset were used for the analysis. Logistic regressions were run both with and without the offset component added to each model. The results for each run for each dataset were collected and averaged. The results across the four Twitter datasets were collected together, and the average was taken across all runs in all four datasets.

**Results.** The impact of adding the offset to each of the models for StackOverflow and Twitter is shown in Figures 15 and 16. In both plots, if the model name includes “full” then all model components are included. When the model components are enumerated, then at least one of the components is left out. The random permutation and Bayesian models are plotted alongside each other so that comparisons across the two model types can be done easily. So, for example, the “standard prior model relaxed across posts without offset” means that only the prior model component was used and the offset for that component was not subtracted off when determining activation. As another example, the “Bayes only orderless context” means that only the context component of the Bayesian model was used (along with the offset), and that no order information was used for that contextual component term.

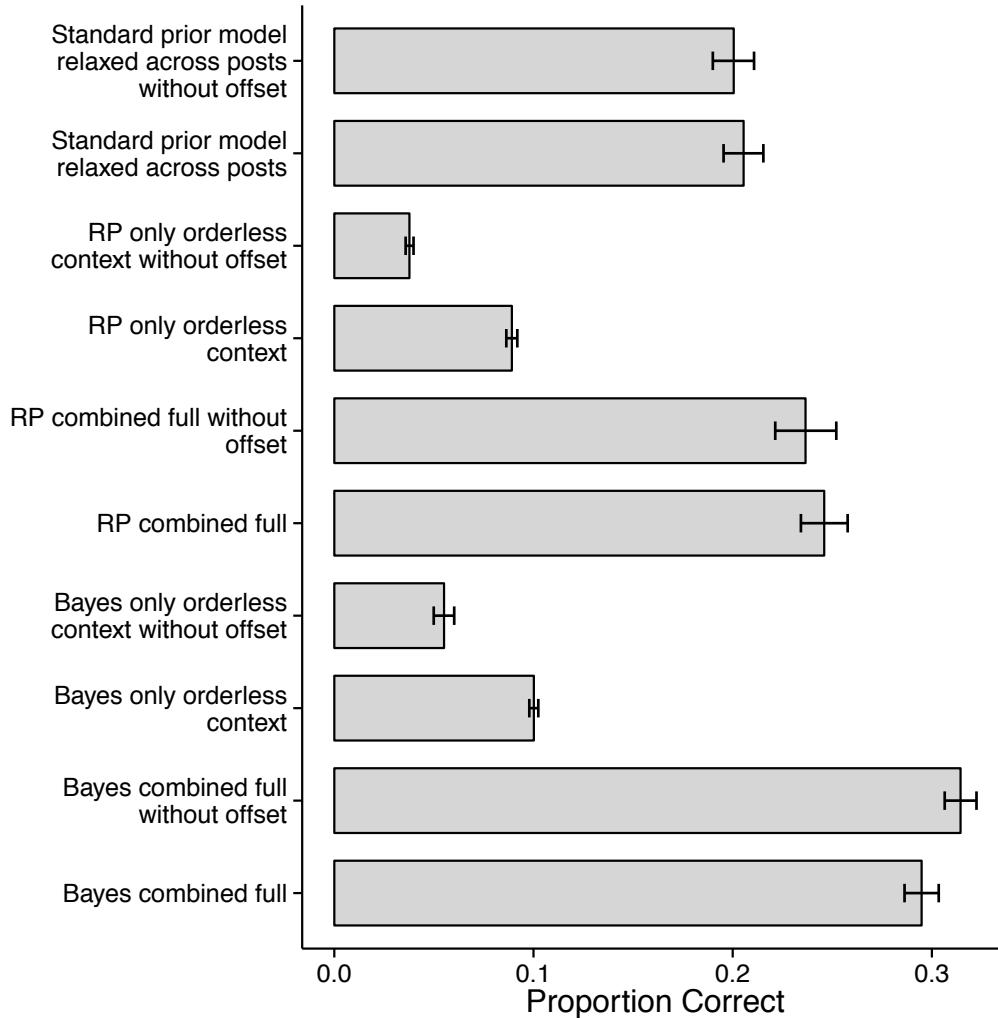
Looking across the plots, including the offset significantly increases model accuracy, for all models, terms, and sites (.53 to .60 and .22 to .45 for Bayesian and random permutation for StackOverflow, .31 to .29 and .24 to .25 for Bayesian and random



*Figure 15.* Impact of adding offset component for StackOverflow. Error bars represent the 95% bootstrapped confidence interval for the mean model accuracy across all runs in dataset. The random permutation models are represented as “RP” and Bayesian models as “Bayes.”

permutation for Twitter), except that the full Bayesian model without an offset was slightly more accurate on Twitter. Accuracy is improved even when only a single term is used. Although in this case the term is not weighted, but using an offset still improves accuracy since the offset term ensures that only a small number of tags per posts are predicted by the model to be the chosen tags.

The random permutation model in particular is greatly affected by the offset term, for both the StackOverflow and Twitter. This suggests that the mean activation for the random permutation model for the top few tags in each post can vary greatly across posts.



*Figure 16.* Impact of adding offset component for Twitter. All error bars are the 95% bootstrapped confidence interval for the mean.

Only if this mean activation is subtracted will the logistic regression model generate optimal weights and predictions that are comparable to the Bayesian model when model accuracy relaxed across posts is examined.

**Discussion.** It is somewhat disappointing that the offset term is even needed. If it were the case that model activation for a tag in a specific post could be interpreted as an exact activation value that did not depend on the specific post, then the model would be able to say when, for example, the third top tag in a post is more likely to be a chosen tag than the second top tag in another post. That is, the activation values would have meaning more than just rank ordering, and could be used to determine, for example, the number of

tags to use for a specific post. If this were the case, then adding the offset term would actually decrease accuracy, since this absolute value of the activation values would be removed. However, since accuracy improves when adding the offset term for both models, it does not seem that absolute activation values have much meaning across posts for these datasets, and that the relative activation values for each tag within a post is what is important.

One simple and likely explanation is that there are posts where a large set of tags have high activation, but the author is choosing to use only a fixed small set of the top tags for each post, regardless of the activation values of the top tags. The author determines the number of tags to use for a post based on competing goals of tagging posts and not overly saturating a post with tags. For example, the author may be writing a question on StackOverflow about *javascript*, and a large set of related tags are retrieved from declarative memory that are all above the retrieval threshold (e.g., *javascript*, *dom*, *svg*, *html*, *jquery*, *css*), but the author only chooses to use the tags *javascript*, *dom*, and *svg*, since he has decided to tag this post with three tags, and those are the three tags with the highest activation. Adding the offset term into the model is a way to implement a simple version of this strategy where the author is actively choosing to use only a small set of tags for each post. Also, this type of strategic behavior of limiting the total number of chosen tags could be easily implemented in the production system of ACT-R.

## Twitter Subsets

As a second methodological concern, model accuracy was examined across each of the dataset subsets collected for Twitter. Four popular-hashtags subsets were collected for Twitter at different time periods. This was to ensure that model results could be interpreted as effects that were not specific to a certain time period or set of popular hashtags.

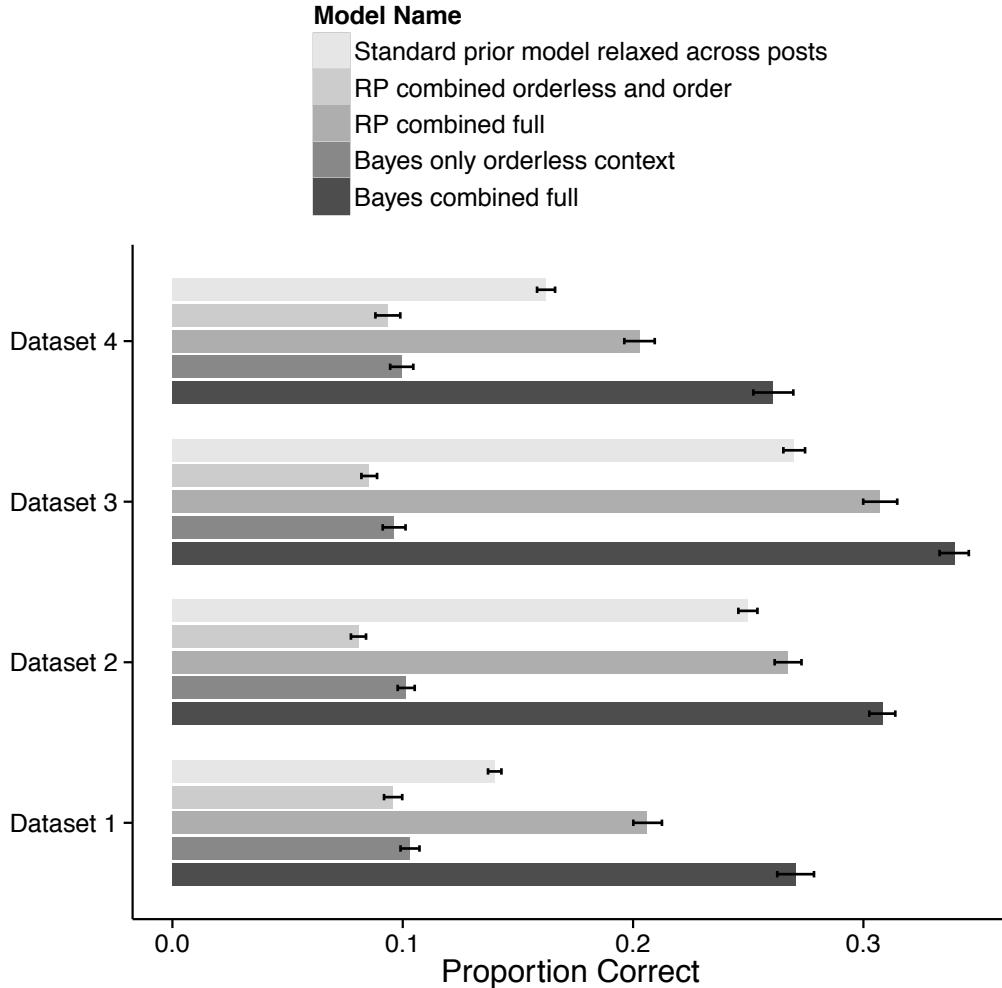
**Method.** The four subsets were collected across a period of one month. Each was collected sequentially, and about one subset was collected per week. The hashtags used in

each subset were chosen by scraping a site that published the current trending hashtags for each city. Before starting to collect for each subset, the current trending popular hashtags for the 400 most populous US cities were collected, and then 400 popular hashtags were randomly sampled from this set. Only 400 were selected since this is the maximum that the Twitter API can monitor at one time.

In order to ensure that model results are not dependent on the specific Twitter subset collected, it is important that the overall accuracy differences between the different models do not drastically shift from subset to subset. That is, it is important that the effect size of model is much greater than the effect size of the interaction between Twitter subset and model. A small interaction relative to the main effect means that the rank ordering of the different models remains consistent across the different Twitter subsets. Further, a small effect size of subset relative to the effect of model means that the exact accuracy for each model remains consistent across the subsets. To measure the effect sizes of subset, model type, and the interaction, a two-way between-subjects ANOVA was performed.

**Results.** Model accuracy results for components of both the random permutation and Bayesian models grouped by Twitter popular-hashtags subset are depicted in Figure 17. Accuracy is affected significantly by subset ( $F(3, 340) = 764, p < .001, \eta^2 = .12$ ), model ( $F(4, 340) = 3617, p < .001, \eta^2 = .77$ ), and the interaction between model and subset ( $F(12, 340) = 142, p < .001, \eta^2 = .09$ ). However, the effect size of model type on accuracy is 8.6 times larger than the interaction term ( $\eta^2 = .77$  versus  $\eta^2 = .09$ ). This shows that the performance of a model relative to the other models remains consistent across the four subsets. Further, the effect size of model type is 6.3 times larger than the effect of Twitter subset ( $\eta^2 = .77$  versus  $\eta^2 = .12$ ). This shows that not only is relative model performance maintained across Twitter subsets, but the exact value of model accuracy for each model remains reasonably consistent across subsets.

**Discussion.** Since the effect sizes of subset and the interaction between model type and subset are minimal relative to the effect size of model, model behavior for the



*Figure 17.* Model accuracy for each of the four Twitter popular-hashtags subsets

popular-hashtags subset for Twitter can be interpreted as behavior that will persist across different time periods, if the same method of choosing popular hashtags is conducted at a later date. It may also be the case that that model behavior does not depend on the specific method of choosing the popular hashtags, but since only a single method was used for this analysis, the results from this study cannot by themselves be used to answer this question.

### Stop Words

The method of handling stop words has been shown to influence model accuracy (Sahlgren et al., 2008; Stanley & Byrne, 2013), so this was the first architectural concern that was examined. Sahlgren et al. (2008) looked at several ways to handle the

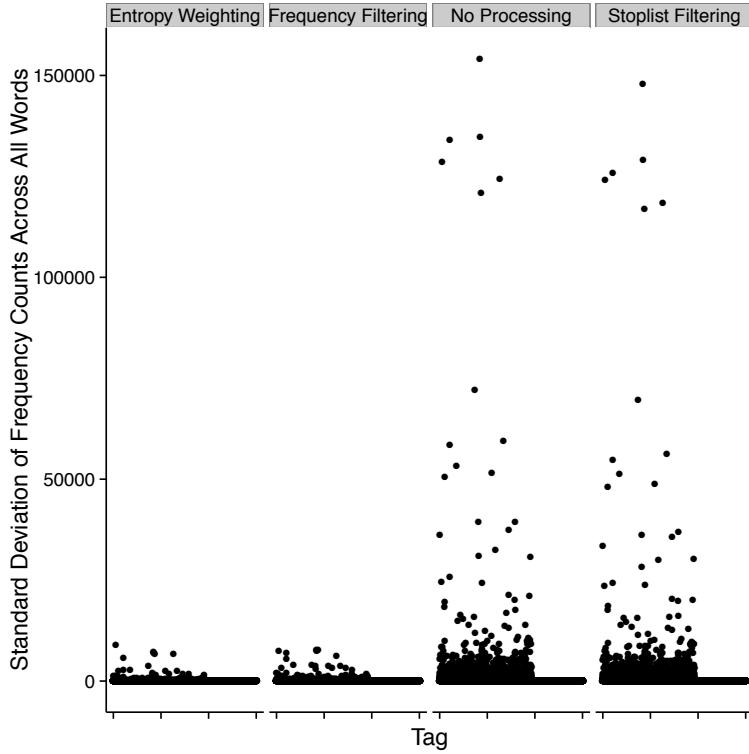
commonly-occurring stop words for the random permutation model: A data-driven frequency-filtering approach was tried, along with using a previously-generated stoplist, and no stop word removal at all. Removing the high-frequency words in the dataset showed the highest performance increase compared to no stop word removal, and using a previously-generated stoplist hardly increased performance. However, Sahlgren et al. did not try to weight the stop words and only tried methods to remove them. So alongside looking at frequency filtering, previously generated, and no stop word removal, Stanley and Byrne (2013)'s method of attenuating stop words based on entropy was also explored.

**Method.** The runs from the Twitter popular-hashtags and StackOverflow randomly-sampled datasets were used for the analysis. Both the random permutation and Bayesian models were tested. Activations for context were computed by using five different methods to handle stop words: The two removal methods used in Sahlgren et al. (2008) (i.e., removal based on frequency, removal based on the pre-determined 571-word Cornell SMART stoplist (Salton & Buckley, 1988)), the entropy-weighting method used in Stanley and Byrne (2013), combining the entropy-weighting and frequency-filtering methods, and no removal or weighting of stop words.

**Determining Frequency Cutoff.** A cutoff point must be identified and used in order to remove the high-frequency words. Sahlgren et al. (2008) used a cutoff where words that occurred more than 15,000 times were removed. Since the StackOverflow and Twitter datasets are much larger than the datasets used by Sahlgren et al., the same cutoff could not be used.

The ideal cutoff was determined by choosing the value where the standard deviation of the counts for each word for each of the tags were all relatively small. The plots for the standard deviations across tags that were used to identify the cutoff for StackOverflow and Twitter are shown in Figures 18 and 19.

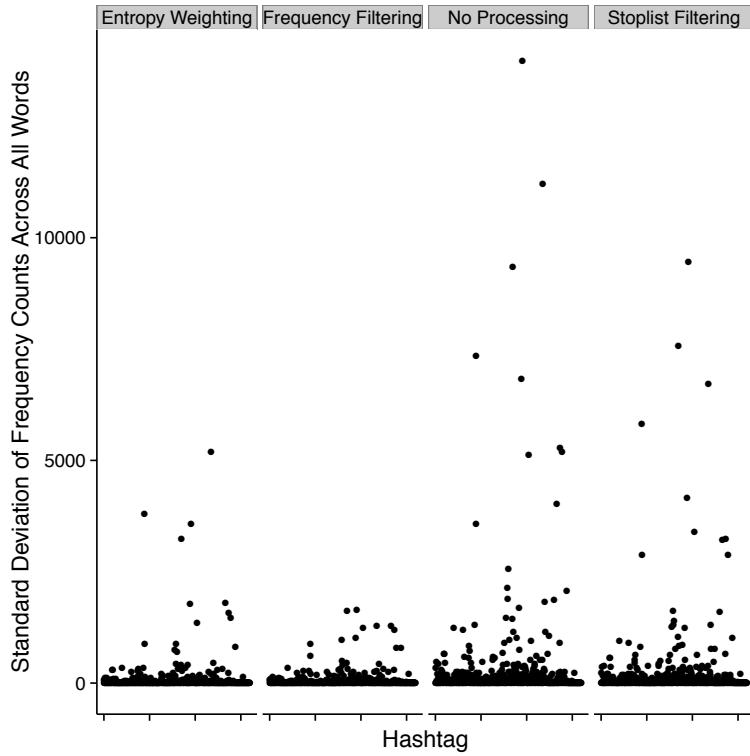
The no processing plot shows the size of the spikes when high-frequency words are not removed. The entropy plot shows how the size decreases by more than an order of



*Figure 18.* Variance in observation counts of words for each tag for StackOverflow. Plots are shown for the four different methods for handling stop words. Each plot contains the standard deviation of the total counts for each word in context as a function of hashtag. High values mean that there are context words for a hashtag that have a high number of counts relative to the counts for all other context words for that hashtag.

magnitude after the entropy weighting measure is used (compare the difference between the no processing subplot and the entropy subplot). The cutoff for the frequency plot was chosen such that the plot looks similar to the entropy plot, with spikes around the same size. The cutoff that produced the frequency plots in Figures 18 and 19 is where a word represents more than 0.04% of total occurrences in the dataset. This may or may not be equal to the value for Sahlgren et al. (2008), since that number was expressed in total observations (not as a percentage), and the total number of words in the dataset was not given.

Note that using a predefined stoplist does not remove some of the remaining high-frequency words in these datasets, which hints that this method may not be ideal when model accuracy is examined. Also, no strong claims are being made that the cutoff

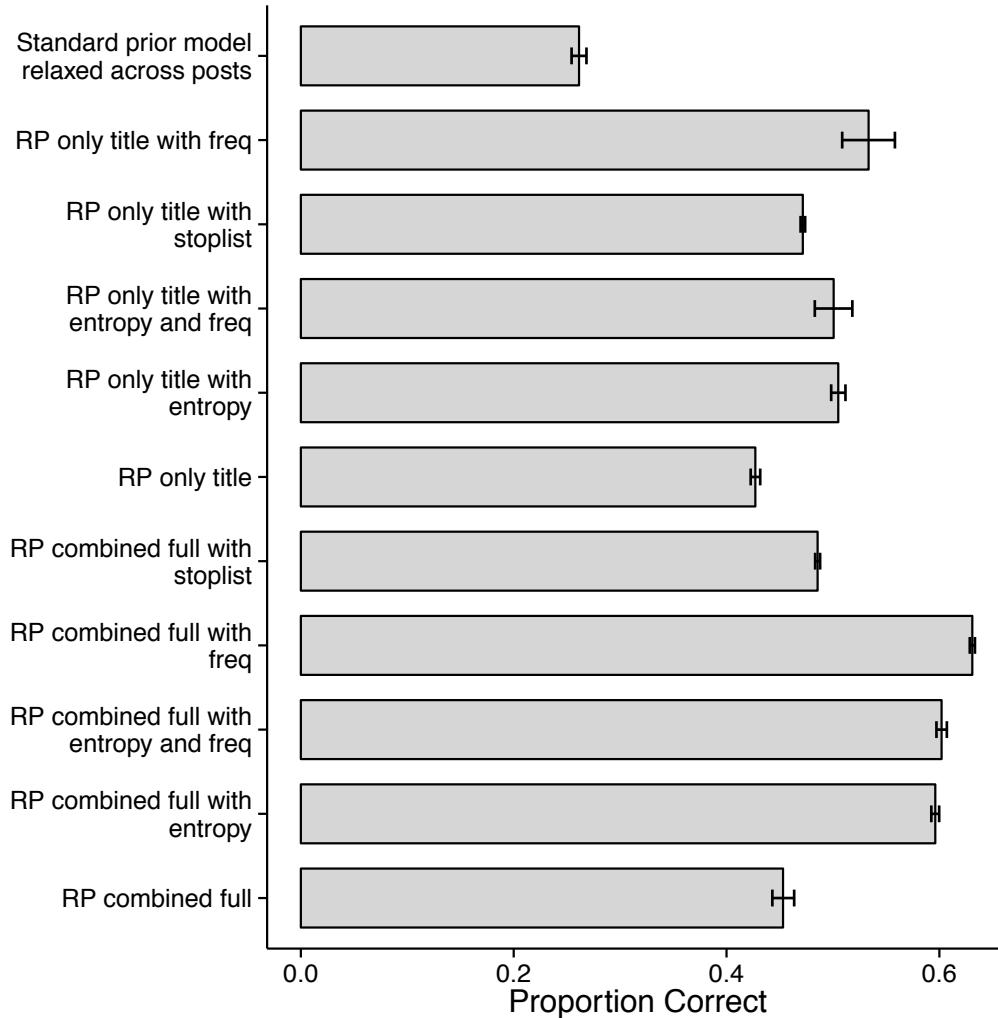


*Figure 19.* Variance in observation counts of words for each tag for Twitter.

identified is the ideal frequency percentage cutoff for all datasets when a frequency cutoff is used. The method used to identify this cutoff was simply intended to find a reasonable value where comparisons between a frequency cutoff and entropy weighting techniques would be equitable.

**Results for the Random Permutation Model.** Plots for the three methods of handling stop words for the random permutation model on the StackOverflow dataset are shown in Figure 20. The results show that model accuracy improves if stop words are handled in any of the three ways (predefined stoplist filtering, entropy weighting, and frequency filtering). Using a predefined stoplist makes the smallest improvement (.45 to .49), while the entropy (.60) and frequency (.63) techniques improve performance the most, and frequency shows a slight edge over entropy. However, using the frequency-filtering technique produces more noise in accuracy measurements compared to the other techniques, at least when tested on words in the title of StackOverflow posts.

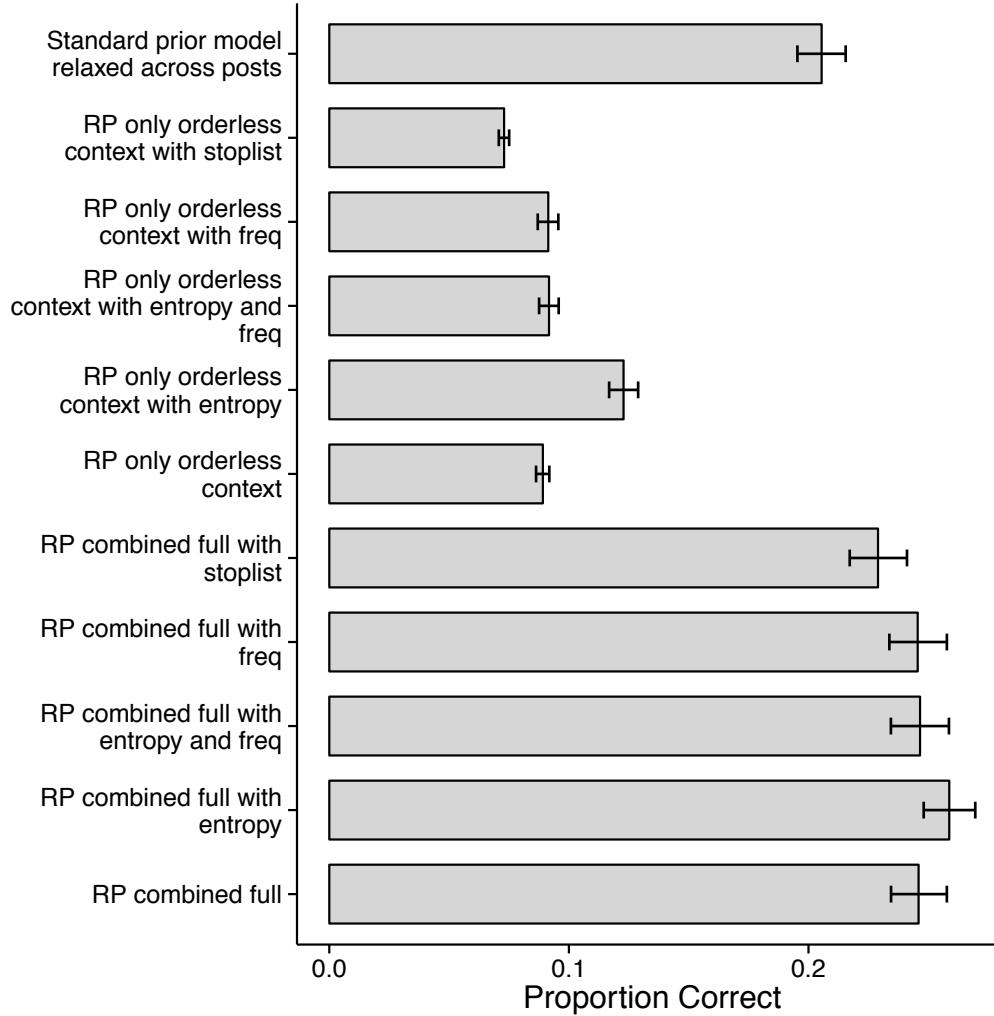
The three stop word techniques were then tested for the random permutation model



*Figure 20.* Stop-word techniques for the random permutation model for StackOverflow. The five methods for handling stop words are shown both for the title model component in isolation and the full model. “Stoplist”, “freq”, “entropy” are the stoplist removal method, removal based on frequency count, and weighting based entropy metric.

on the Twitter dataset, and the results are depicted in Figure 21. For the random permutation model for Twitter, the entropy weighting technique is the best method for handling stop words (.25 to .26). Removing stop-words based on a predetermined list actually decreases accuracy (.25 to .23), and using data-driven frequency filtering has only a small to no effect (.25 to .25).

**Discussion.** Comparing across Figures 20 and 21, entropy and frequency techniques are superior to filtering based on a predetermined list. However, it is more difficult to choose the best approach when comparing entropy weighting and frequency filtering.



*Figure 21.* Stop-word techniques for the random permutation model for Twitter. The five methods for handling stop words are shown both for the context model component (i.e., no prior term) and the full model.

Frequency filtering is slightly better for StackOverflow, while entropy weighting is slightly better for Twitter. However, frequency filtering does have a few drawbacks compared to entropy weighting: It produces accuracy measurements that have higher levels of noise (see Figure 20 and the frequency model for the title component in StackOverflow). Also, the ideal cutoff threshold must be identified in order to filter based on frequency, and identifying this cutoff requires a search, and this search must be done for each dataset that frequency filtering is used on. This cutoff value is a free parameter in frequency filtering that is not required when weighting stop words by entropy. Therefore, favoring parsimony,

for the random permutation model, attenuating stop words based on entropy is a slightly superior method compared to removing stop words based on frequency of use. This result is interesting since filtering stop words based on frequency is the more common default method for random permutation models (Sahlgren et al., 2008). This suggests that a simpler method (entropy weighting) can be used instead of the more common frequency-filtering technique to achieve the same (or better) accuracy measurements.

**Results for the Bayesian Model.** Since accuracy measurements when using a predetermined list to remove stop words for the random permutation model were much worse than the frequency and entropy methods, this method was not examined for the Bayesian model. The results for the frequency and entropy techniques for the Bayesian model on the StackOverflow dataset are depicted in Figure 22.

The results show that entropy weighting produces the most accurate results (.61 to .63). Also, accuracy actually decreases when using frequency filtering (compared to no filtering) (.61 to .56) for the Bayesian model. So entropy weighting is the clear winner in this case.

Similar results are found when the Bayesian model is tested on the Twitter dataset. Figure 23 shows that entropy weighting is again the best performer (.29 to .32), and frequency filtering decreases performance relative to no filtering (.29 to .28).

**Discussion.** One result that should not be overlooked is how poorly using a predetermined stoplist performed when compared to the two data-driven frequency and entropy approaches. This is most likely because the StackOverflow and Twitter datasets are large and contain high-frequency words that are domain specific (e.g., emoticons) and are not included in the predetermined list. This shows that data-driven techniques to identify stop words can be much more effective than using a predetermined list.

Another interesting result is how the Bayesian model actually performed worse when stop words were removed based on frequency compared to no stop word removal. This was not the case for the random permutation model in this study, was not the case when

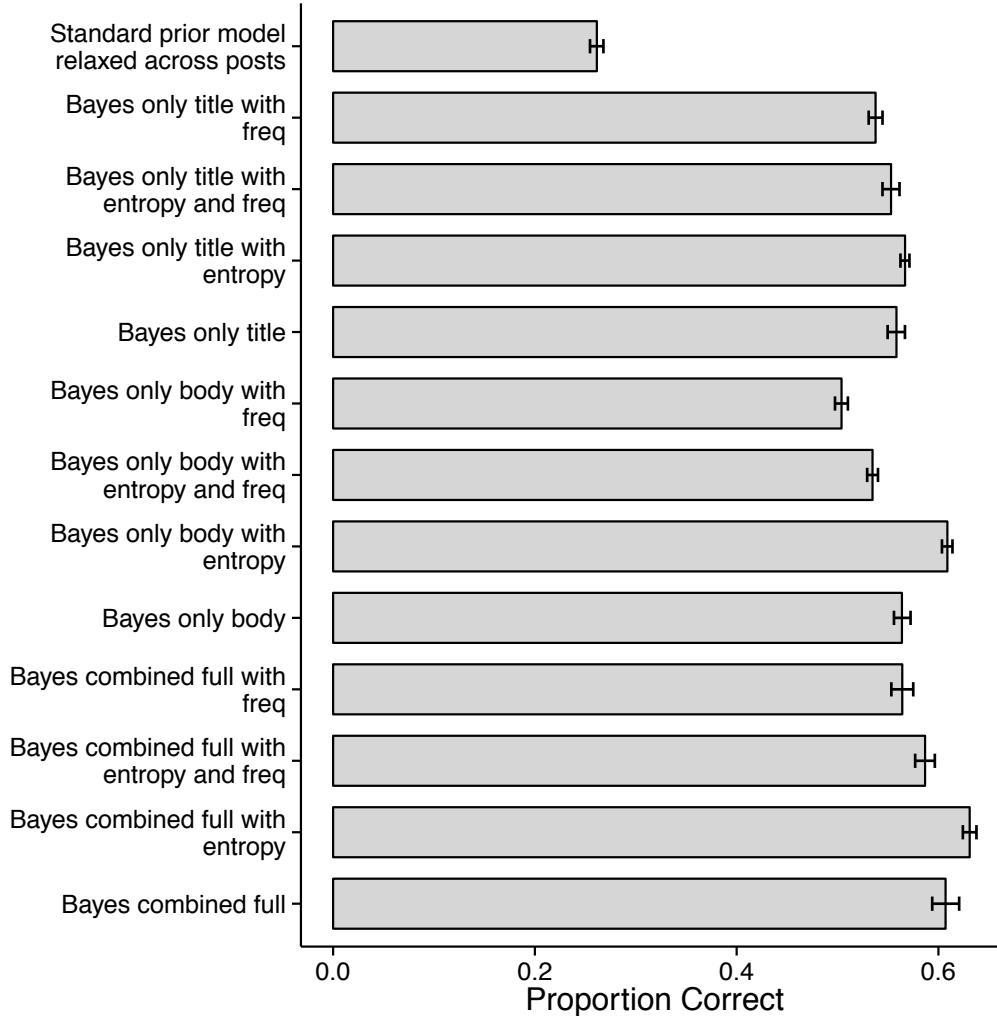


Figure 22. Stop-word techniques for the Bayesian model for StackOverflow

Sahlgren et al. (2008) used frequency filtering on the random permutation model, and is most likely never the case for the random permutation model. Model accuracy most likely decreases for the Bayesian model because this model already has a normalization component built in that the random permutation model does not have. When computing activation for the random permutation model, the correlation is calculated directly on a matrix of counts of co-occurrences for words and tags. For the Bayesian model this count matrix is converted to a log-odds ( $S_{ji}$ ) matrix, where both the number of observations for the word and the tag are normalized when computing the value for each cell (see the  $S_{ji}$  computation in Table 3). So it is likely the case that a form of frequency attenuation is

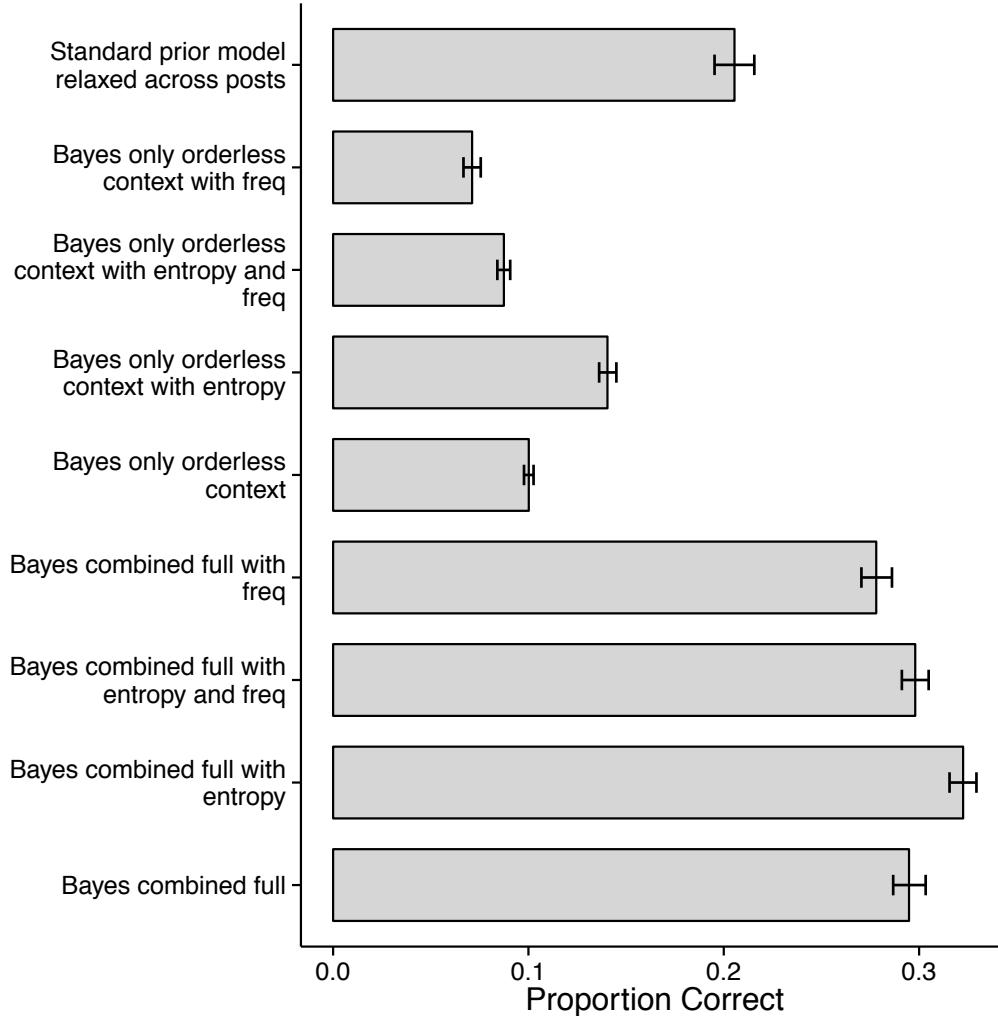


Figure 23. Stop-word techniques for the Bayesian model for Twitter

already built into the Bayesian model. Therefore, adding a frequency filter on top of this does nothing to improve accuracy and only removes predictive information.

Stanley and Byrne (2013) found that the entropy-weighting technique to attenuate stop words for the Bayesian model significantly improved model accuracy. The same effect is found for the StackOverflow and Twitter datasets in this study. Further, when this entropy-weighting technique is compared directly to the more common frequency-filtering technique, weighting by entropy produces more accurate results. This was clearly the case for the Bayesian model. Both frequency and entropy techniques performed equally for the random permutation model. However, since the entropy-weighting technique is parameter

free and does not require a cutoff to tune, it is also a better choice for the random permutation model for these datasets.

The effect sizes when comparing no stop-word removal technique to either the entropy or frequency techniques are quite large. This shows the importance of correctly identifying and handling stop words when building co-occurrence matrices for large-scale natural language datasets. For further research, it may be worthwhile to explore an even larger range of stop-word handling techniques, since model accuracy seems to be quite sensitive to the specific method used. However, one conclusion that can be made from this research is that the entropy-weighting method is a contender, and should be considered much more often than it has been previously.

## Corpus Size

The number of documents used to build the co-occurrence matrix has been shown to influence model accuracy (Sahlgren et al., 2008), so this was the second architectural concern that was examined. The default number of documents used to build the co-occurrence matrix was 1,000,000 for StackOverflow and 3,000,000 for Twitter. In order to see how accuracy changed as function of corpus size, co-occurrence matrices for a range of smaller corpus sizes were built. Both the random permutation and Bayesian models were tested on all corpus sizes.

**Method.** For StackOverflow, the following number of posts were used to build separate co-occurrence matrices: 1000, 10,000, 100,000, and the default 1,000,000. For Twitter, these matrix sizes were tested alongside the default 3,000,000. All runs from the four Twitter popular-hashtags datasets and the randomly-sampled StackOverflow dataset were used to test model accuracy.

**Results for StackOverflow.** Model accuracy as a function of co-occurrence matrix size for StackOverflow is depicted in Figure 24. Accuracy improves for both the Bayesian and random permutation models as the size of the documents used to generate the

co-occurrence matrix increases. However, model accuracy using the entropy weighting technique plateaus and slightly decreases for the largest corpus size for the random permutation model, while the frequency technique only begins to plateau. The Bayesian model does not plateau and accuracy continues to improve as corpus size increases.

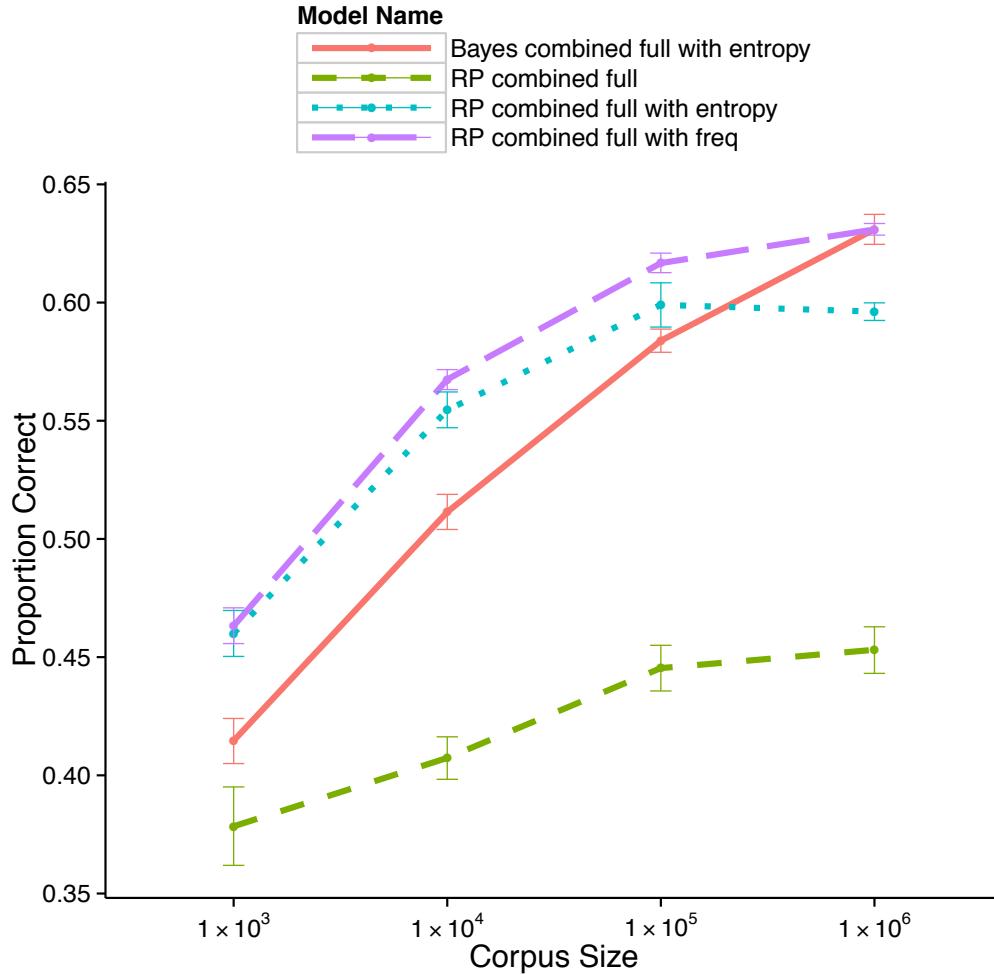


Figure 24. Effect of size of co-occurrence matrix for StackOverflow. Error bars represent the 95% bootstrapped confidence interval for the mean model accuracy across all runs in dataset.

**Discussion.** It is likely that the random permutation model is starting to saturate when entropy weighting is used for the largest corpus size for StackOverflow. Saturation happens when certain words co-occur with a tag much more often than all of the other words. This introduces range effects when the correlation is computed, which ends up driving all of the correlations near one, and consequently decreases discrimination and

accuracy. Saturation happens when entropy weighting is used because the non-predictive words are only attenuated and not removed, and this attenuation is not enough to overcome the range effects that grow with corpus size.

However, the effect of saturation is small compared to the large effect from using either frequency filtering or entropy weighting to handle the stop words. The Bayesian model does not saturate when using entropy weighting because each cell in the co-occurrence matrix is normalized by word frequency and tag frequency.

**Results for Twitter.** The overall trend of increasing accuracy with corpus size is present for Twitter as well. Both entropy weighting and frequency filtering in Figure 25 plateau for the random permutation model with increasing corpus size. Once again, the Bayesian model continues to improve with corpus size.

**Discussion.** It is interesting to compare model accuracy between the Bayesian and random permutation model at the smallest and largest corpus sizes for both StackOverflow and Twitter. At smaller corpus sizes, the random permutation model outperforms the Bayesian model (slightly for Twitter, dramatically for StackOverflow), and this performance difference deteriorates as corpus size increases. In other words, the Bayesian model needs larger co-occurrence matrices than the random permutation model to work properly. As the size of the dataset increases, the compression may start to lose crucial information, and the uncompressed Bayesian model should be used. As the size of the dataset decreases, the noise and crosstalk introduced by the compression for the random permutation model is actually beneficial, and the compressed random permutation model should be used. This noise is beneficial because it helps soften spurious  $S_{ji}$  values when the counts for each individual cell in the co-occurrence matrix are still volatile, which happens at smaller corpus sizes.

This result has important implications. The corpus size is rarely chosen, and more likely constrained by the number of documents in the dataset being studied. If that dataset is relatively small, it may be better to use the compressed random permutation model than

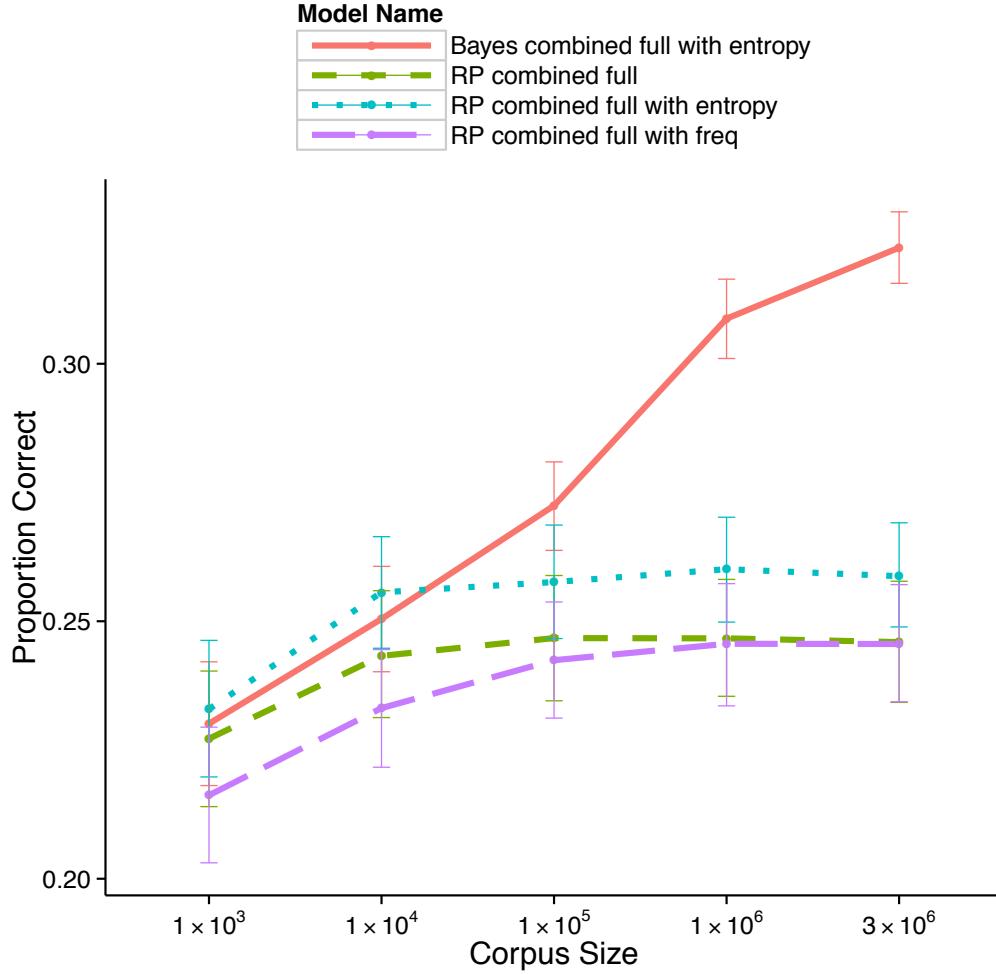


Figure 25. Effect of size of co-occurrence matrix for Twitter. Error bars represent the 95% bootstrapped confidence interval for the mean.

the Bayesian model, and vice-versa if the dataset is large.

However, corpus size is not the only factor that determines whether the Bayesian model or random permutation model will perform better. For example, the corpus size where the Bayesian model begins to outperform the random permutation model is quite different for StackOverflow and Twitter. This is because the number of observations used to generate the co-occurrence matrix is a function of four components: corpus size, number of words in each document, total number of unique tags, and number of tags used per document. The average number of tags used per post for StackOverflow and Twitter were the same order of magnitude (around 3 for StackOverflow, slightly less for Twitter).

However, the average number of words in a StackOverflow post is much greater than the 140 character limit used for tweets. More words per post means that the number of observations per cell in the co-occurrence matrix grows more quickly with corpus size. The Bayesian model is more likely to perform better with datasets like StackOverflow that have these larger corpus sizes.

Also, the popular-hashtags Twitter dataset only contains 400 unique tags while the randomly-sampled StackOverflow dataset has 34,377 unique tags. More unique tags means that the co-occurrence counts are spread across a wider range of tag columns, which means that it takes a larger number of total documents before the Bayesian model performs better than the random permutation model.

The transition point where accuracy for the Bayesian model outperformed the random permutation model was much higher for StackOverflow than Twitter. This is most likely because StackOverflow has almost 100 times more unique tags than the Twitter popular-hashtags dataset. The number of words in a StackOverflow post is also much higher than Twitter, which should have lowered the transition point. However, the effect of words in a post must be less than the effect of number of unique tags, as the transition point was still much higher for StackOverflow given that it should have lowered due to post size.

The compressed random permutation model should perform better with smaller corpus sizes, less words per document, more unique tags, and fewer number of tags per post. However, since there are so many factors that determine whether the Bayesian or random permutation model will perform better, and only two different datasets were examined in this research, it is difficult to make any specific recommendations as far as exact transition cut points (e.g., corpus size, words per document) where either the Bayesian or random permutation model should be used. Also, it is likely that the specific domain being studied interacts with the factors already mentioned to determine which model is better for that domain. Nevertheless, if a researcher is working in a domain with limited data (e.g., small corpus size, large tag space) that produces a sparse co-occurrence

matrix, then it may be worthwhile to experiment with using a compressed random permutation model instead of the full Bayesian model.

### Amount of Compression for Random Permutation

Another parameter in the random permutation model is the number of rows used to represent all of the different words. Each word's environment vector is represented by two randomly placed ones and two negative ones across the rows. All that is required is that the four positions and signs of each of these environment vectors is unique. But this means multiple environment vectors can overlap when looking at a single row. This is a lossy form of compression, and allows the number of rows to represent the words to be much less than the total number of words. As the number of rows decreases, the amount of cross talk between the environment vectors increases, compression increases, and the amount of information stored decreases. As a third architectural concern, the effect of compression for the random permutation model was examined for Twitter and StackOverflow.

**Method.** Three different row-size values were tested for the random permutation model to see how accuracy changed as a function of space required to represent the co-occurrence matrix. A low value (100), a more standard value (2048), and a high value (10,000) of rows were tested. A separate co-occurrence representation was built for each of these row values. Each representation was tested on the four Twitter popular-hashtags datasets and the StackOverflow randomly-sampled dataset. Accuracy measurements were the same as were used for previous runs of these datasets: mean accuracy for each of the 5 StackOverflow and 15 Twitter runs.

**Results.** Results for using different levels of compression for the random permutation model for the StackOverflow dataset are depicted in Figure 26. In the figure, the model name that does not have the number of rows in the matrix specified is the 2048-row matrix. This was done so that names in all of the plots would remain consistent across all plots, and the 2048-row matrix is the size used for the random permutation

model for all other results. The plot is also broken out for different size datasets used to build the co-occurrence matrix.

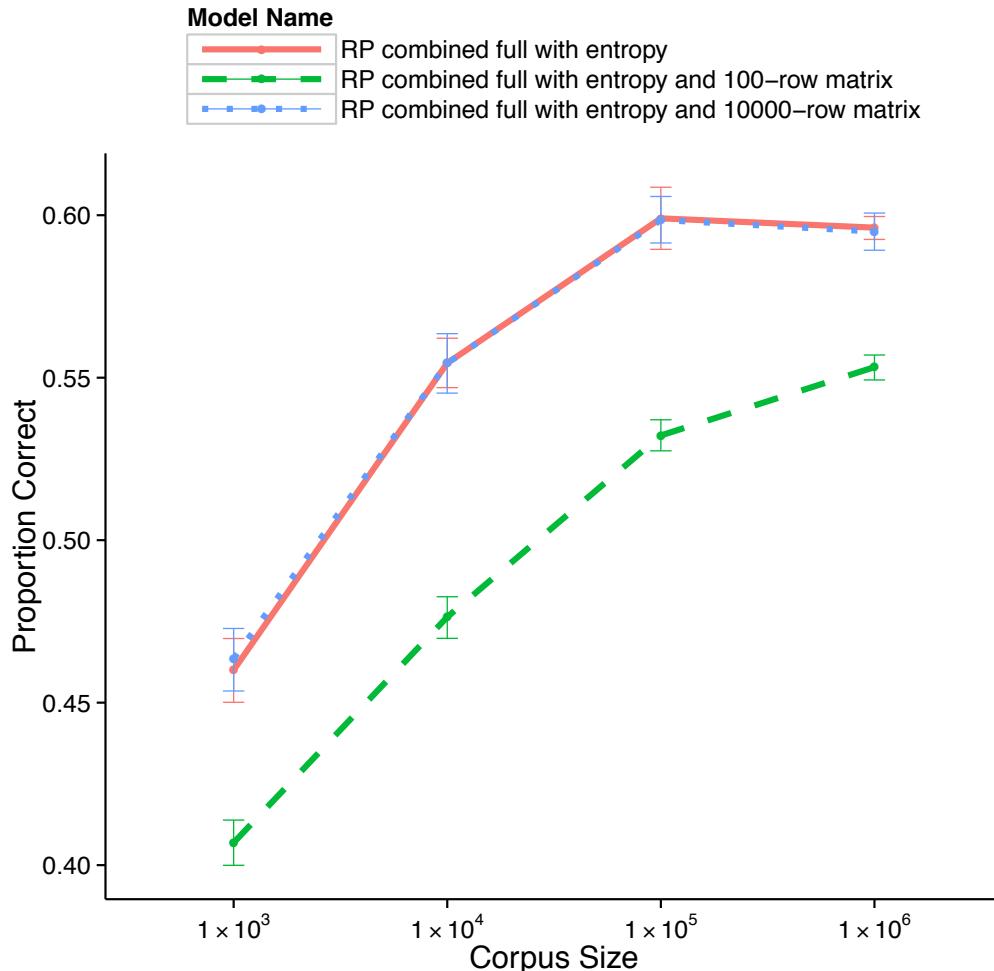


Figure 26. Effect of compression on random permutation model for StackOverflow. Error bars represent the 95% bootstrapped confidence interval for the mean.

Reducing compression to only 100 rows in the representation has a strong negative effect on accuracy (.60 to .55 for the 1,000,000 document corpus size). But 2048 rows is sufficient for these datasets, as increasing the number of rows to almost an order of magnitude higher (10,000) has little to no positive impact on accuracy (.60 to .60 for the 1,000,000 corpus size).

Results are similar for Twitter. Figure 27 shows the same negative impact when reducing from 2048 to 100 (.26 to .24 for the 3,000,000 corpus size), and little to no impact

when increasing from 2048 to 10,000 (.26 to .26 at 3,000,000). These results are consistent across all corpus sizes used to build the co-occurrence matrix.

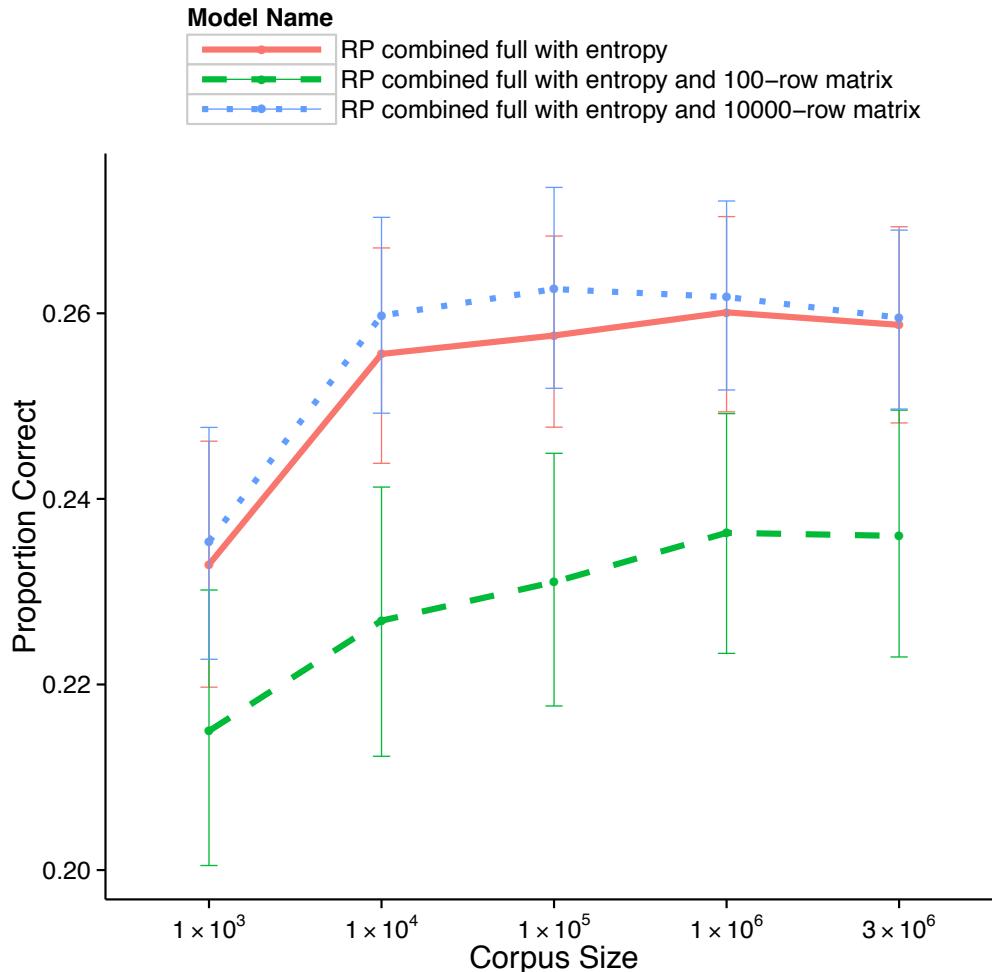


Figure 27. Effect of compression on random permutation model for Twitter. Error bars represent the 95% bootstrapped confidence interval for the mean.

**Discussion.** The fact that the random permutation model only needs around 2048 rows to represent all of the words in the Twitter and StackOverflow datasets is impressive. Twitter has around 2,383,000 unique words for the 3,000,000 tweets in each of the popular-users datasets. StackOverflow has 5,848,000 unique words for the 1,000,000 posts used to generate the co-occurrence matrix. So it is impressive that only 2048 rows are needed to represent most of the variance from all of the words in these co-occurrence matrices. This is likely because word use frequency for StackOverflow and Twitter

approximately follows Zipf's law. Stanley and Byrne (2013) showed that frequency of tag use for StackOverflow approximately follows a Zipf's law distribution. This means that if the words are rank sorted by number of times they are used, the frequency of word use drops off sharply when moving to words that are used fewer times. Since these words have little to no ability to influence predictions, it makes sense that they can be safely removed from the dataset.

However, it should be pointed out that the way the random permutation model compresses the words is completely random. It is not an LSA approach where the few ones and negative ones (i.e., the factors) for each word are chosen such to maximize the ability for the compressed matrix to represent the full matrix. The environment vector for each word is randomly selected, and no SVD is performed to choose these in a more discriminating manner. So it is that more impressive that even with a randomly-chosen form of compression, only 2048 rows are needed to represent all of the words in these datasets.

## Combining Prior and Context

The activation value for the context component of the vector-based model is a correlation, ranging from zero to one. This is not a log-odds value. For the previous results, prior and context for the random permutation model were combined by simply adding the terms. However, this means that ACT-R's prior base-level learning component (which is a log-odds value) is added to a correlation value, which may not be mathematically appropriate. As a fourth architectural concern, another method to combine the prior and context components for the random permutation model was explored.

**Method.** For a given retrieval, the random permutation model returns a set of activations that are correlation values for each of the tags. Instead of simply adding this correlation value to the prior model term, it may be more appropriate to convert the correlation value to a log-odds value first, and then add this context term to the log-odds

prior term. So a method was created to convert a distribution of correlations to a distribution of log-odds values. To do this, the one-tailed cumulative distribution function value for each correlation in the distribution was computed. That probability was then converted to a log-odds value in the usual way.

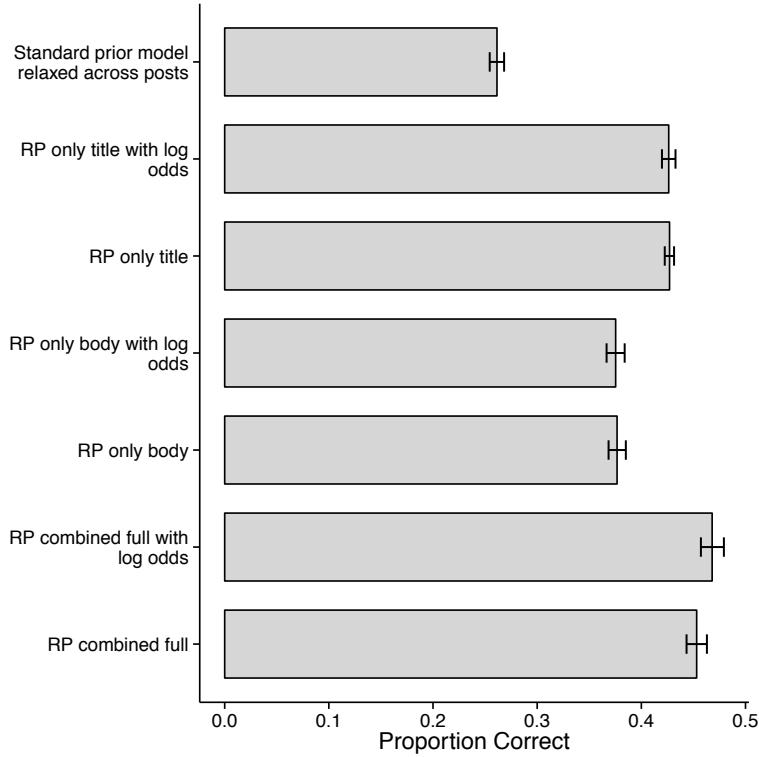
For example, say that the model is choosing between a set of tags, and those tags have a mean correlation of .2 with context and a standard deviation of 0.1. If a tag in the set has correlation of .2 with context, then approximately half of the observed tags have a correlation lower than this tag. So the one-tailed cumulative distribution value for this tag is .5. If a tag in the set has a correlation of .3, then approximately half plus .34 (cumulative distribution of normal curve from mean to 1 standard deviation outward) of the observed tags have a lower correlation. So the one-tailed cumulative distribution value for this tag is .84.

This technique has a few advantages: It is simple and computationally cheap. Also, it behaves properly at specific points in the distribution. For example, if the conversion is performed on the mean correlation value in the set, the result is a log-odds value of zero. This makes sense given that log odds can be interpreted as the amount of odds information (positive or negative) that should be adjusted to the prior log odds, given that new piece of information. Since the mean correlation value contains no information regarding if it is better or worse than the other correlation values, it is appropriate to assign this correlation value a zero log-odds adjustment value.

The log-odds transformation technique was tested for the random permutation model on the Twitter popular-hashtags and StackOverflow randomly-sampled datasets.

**Results.** Model accuracy results after using the log-odds transformation are depicted in Figures 28 and 29. Using the log-odds transformation technique has a small effect on model accuracy for StackOverflow (.45 to .47) and minimal to no effect on Twitter (.25 to .25). When the technique is used and accuracy is examined for a model with only a single predictor, the transformation does not change accuracy. When predictors are combined and

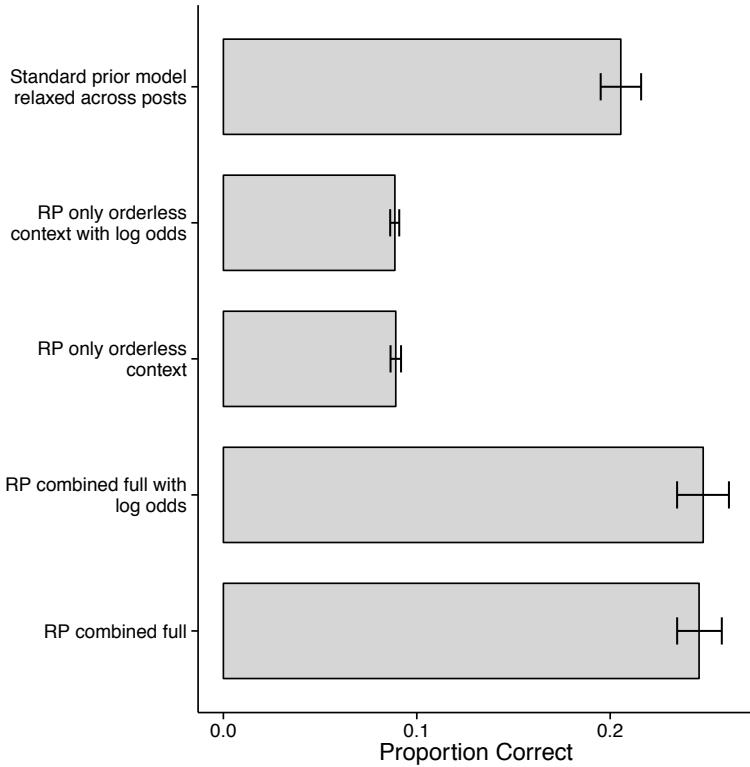
the full model is examined, the transformation is in the direction of slightly increasing accuracy, at least for the StackOverflow dataset. However, any improvement when using the transformation is minimal.



*Figure 28.* Random permutation model accuracy when using log-odds transformation on context for StackOverflow

**Discussion.** Vector-based models such as BEAGLE (Jones & Mewhort, 2007) and the random permutation model (Sahlgren et al., 2008) have included only a context component in the past (i.e., no prior component). When the prior component is included for the random permutation model, accuracy improves significantly (compare accuracy for full models to context models in Figures 28 and 29). However, it is somewhat surprising that it does not matter much how the context and prior terms are added for the random permutation model. Model performance is similar when the context component used for the model is based on a correlation compared to a log-odds value.

There may be a slight edge in increased accuracy when a log-odds transformation is used for the StackOverflow dataset. However, that result is not statistically reliable for the



*Figure 29.* Random permutation model accuracy when using log-odds transformation on context for Twitter

sample size used in this analysis (confidence intervals overlap in Figure 28), and even if present the overall change in accuracy is small (.45 to .47). Nonetheless, it does seem appropriate to convert the context to a log-odds adjustment value when context is added as an additional term to the random permutation model. Since performance does not decrease when the transformation is made, may slightly increase, and leads to a more natural interpretation of each model term as a log-odds value, it is reasonable to include this step for the random permutation model. Future research could explore additional methods to properly combine the terms.

## Word Order

As a final architectural concern, vector-based models with and without word order were tested to examine the effect of word order on model accuracy. Vector-based models like the random permutation model can easily incorporate word order when making

predictions. This is one of their primary strengths that has been discussed (Jones & Mewhort, 2007; Sahlgren et al., 2008). In order to thoroughly test the impact of adding word order into the model, various methods for incorporating word order were tested on the StackOverflow and Twitter datasets.

**Method.** Three different methods for incorporating word order in the random permutation model were tested: taking just the sign of the word's position relative to the tag (direction method), using the position (standard method), and using the position but only including words that were used near the tag (window method). For the standard method, all words and their respective position are used to build the representation. For the direction method, essentially two representations are created: all co-occurrences with words to the left of the tag, and all co-occurrences with words to the right of the tag. The window method is similar to the standard method, where each word's relative position is maintained, but it only includes words near the tag to build the representation. Sahlgren et al. (2008) tested similar methods, and found that accuracy was highest when a narrow window was used for the window method (only including words at most two positions to the left and right of the tag). The same  $+2 - 2$  range was used to test the window method on the Twitter dataset.

Only the Twitter dataset was tested, since the StackOverflow dataset contains no word order information between words in the title and body of the post and the tags used for the post. The author chooses tags after filling out the title and body forms in the post, so the tags are never intermixed with words in the post like they are for Twitter.

The same four Twitter popular-hashtags datasets were used for testing. Both an ordered and orderless context component were created for the random permutation models. The orderless context component uses the same co-occurrence matrix used by the Bayesian model to create the representation, since the Bayesian model does not contain any order information. Models were tested that used only one of the three word order methods for the ordered context component, used only the orderless context component, used a

combination of the two components, and combined context with prior.

**Results.** Model accuracy results after incorporating word order into the random permutation model are depicted in Figure 30. All bar plots that include an order component where “window” or “direction” is not included in the title use the standard ordering method (no direction or windowing, all position information is included). This is the default ordering method used, so the names for these bar plots were kept consistent to the names used in other figures so that comparisons could be made across figures if necessary (similar to what was done for the default row size for the random permutation model). Base models that do not use entropy weighting were also included so that effect sizes could be compared between entropy weighting and adding word order information.

Looking first at models where no prior is added, the random permutation model using only orderless context performs similar to the model using only order context with direction. All other order-only models perform worse than the order model using direction and worse than the orderless model.

Full models were also plotted against the model where only the prior and orderless context was included (i.e., no order). This was to test how much of the variance in model accuracy is shared between the orderless and order models. The results show that there is little unique variance in the order-only models, so order information contributes little to overall model performance above and beyond what an orderless model contributes. This can be seen by comparing the two-component model with only prior and orderless context to the three-component model with prior, orderless, and any of the three methods for the order context term.

**Discussion.** This was a surprising result. One of the main benefits discussed for the random permutation model is that it can easily incorporate word order information into the representation. But including word order information does not dramatically improve performance. For the random permutation model, it is much more important to properly handle low-predictor stop words and include a prior component than it is to include word

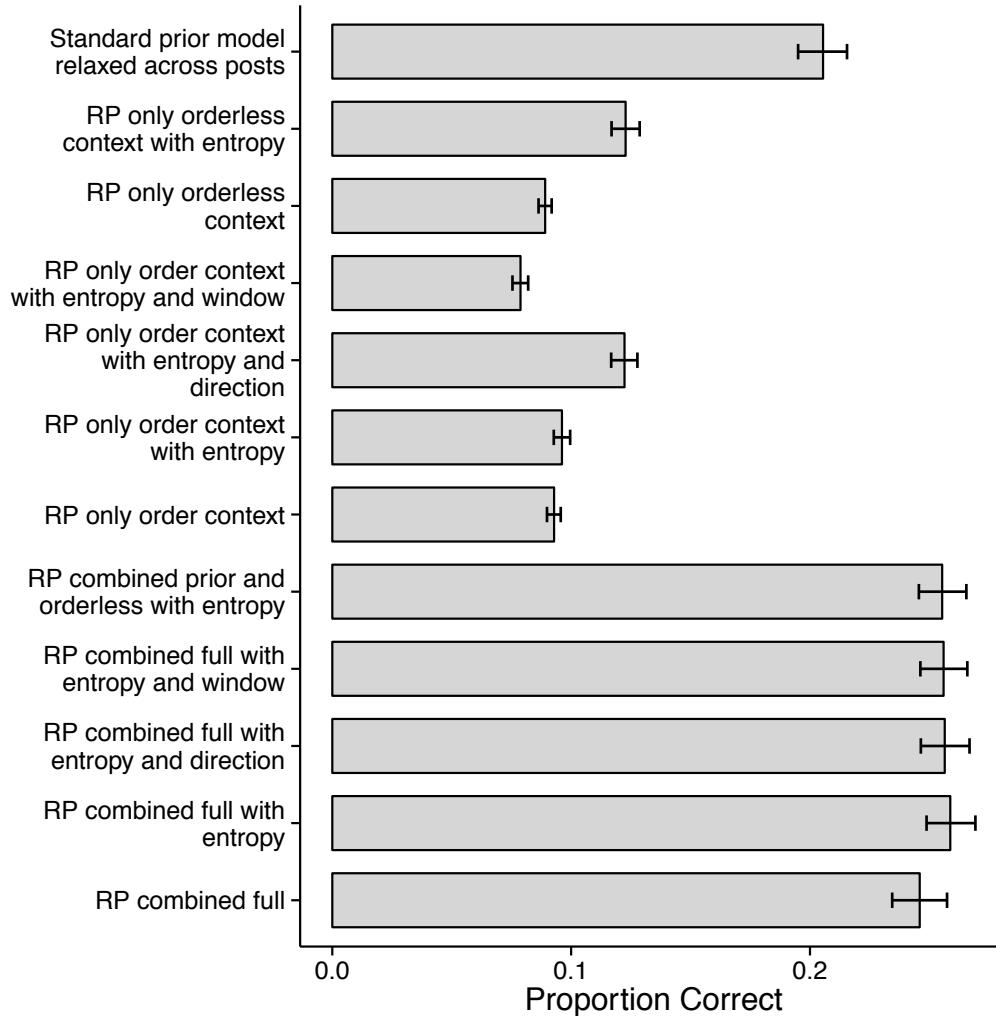


Figure 30. Model accuracy for various word order methods for the random permutation model for Twitter

order information, at least for the Twitter dataset tested in this study.

The original BEAGLE model (Jones & Mewhort, 2007) actually showed only minimal improvement when incorporating word order as well. In absolute terms, this model only improved by 2 percentage points in accuracy when trained on the TASA and tested on the TOEFL. Sahlgren et al. (2008)'s results for word order for the random permutation model are similar. At a row size of around 2000, model accuracy improves by less than a percentage point when order information is incorporated. The results for larger row sizes are somewhat inconsistent, since the full model (order and orderless) sometimes underperforms the order-only model, which should never happen since the order-only

model is a subset of the full model. Sahlgren et al. (2008) most likely did not use a regression technique to optimize the weights for each of the order and orderless model terms. So although incorporating word order into the random permutation model does seem to slightly improve performance, the strength of the model is more likely to be in how efficiently it can represent the co-occurrence matrix rather than its ability to include word order in the representation. That is, the strength of the model is in its compressed unordered context component term and not the ordered component, at least for the Twitter dataset studied.

Also, the reason why the directional version of ordering is the best predictor of the three ordered components is most likely because order information does not have a large effect, so the two representations for words to the left and right of a tag are highly correlated. This means that this directionally-ordered component is essentially composed of all the co-occurrence observations in the orderless component after being randomly sampled and partitioned into two separate representations. The co-occurrences in these representations are highly correlated with each other, and highly correlated with the orderless component, which is why model accuracy is similar for the orderless component and direction ordered component, and why accuracy does not improve when adding the direction ordered component to the orderless component.

Since it was shown that including word order information has little to no significant improvement on model accuracy, no attempt to incorporate word order information into the Bayesian model was made.

## Coefficients for Predictors

All of the methodological and architectural tests were done on the Twitter popular-hashtags dataset and the StackOverflow randomly-sampled dataset. For these tests, a logistic regression statistical technique was used to determine the optimal weight for each model component. A regression was done for each run in the dataset, and since

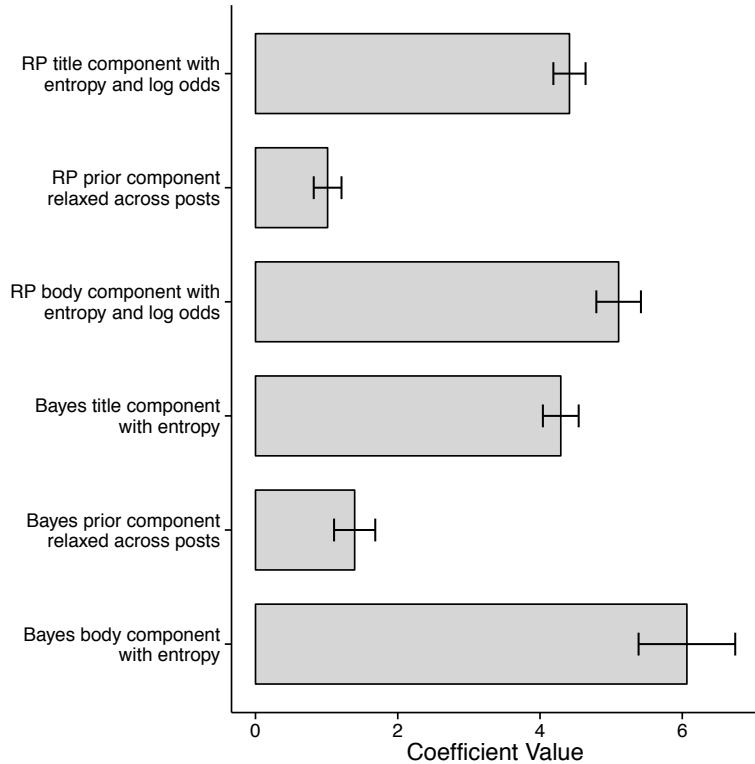
multiple runs were used for the StackOverflow and Twitter datasets, multiple samples of coefficient weights for each model component were computed. The variance of the weights for each component show how likely it is that the regression is overfitting the data, and the mean shows the best estimate weight of each component relative to the other.

**Method.** Coefficients for all of the runs for the Twitter popular-hashtags and StackOverflow randomly-sampled datasets were collected. This totaled 5 sets of coefficient weights for StackOverflow (5 runs) and 60 for Twitter (15 runs for each of the 4 dataset samples).

**Results.** Coefficients for each model component of the full models are depicted in Figures 31 and 32. The range of the distribution of coefficient values within a component is much more narrow than the range of the mean coefficient values between the components of each model. This shows that the model is not being overfit, as the weights for each component are relatively stable across the different runs.

The weight of the prior relative to the context terms is also fairly consistent across models and across the two sites. Across the sites, context is a much stronger predictor than past behavior for StackOverflow, but past behavior is more important for Twitter. Within a site, both models are consistent in how they favor context or past behavior. Also, looking at StackOverflow, both models are consistent in weighting the body more than the title, and the title more than the prior term.

**Discussion.** It was not surprising that the model was not overfit in this scenario, since the number of observed tags is much greater than the number of coefficients being fit. Each model has at most three terms, those weights are being fit to 500 posts within each run for Twitter and 100 for StackOverflow, and multiple tags are used for each post (particularly for StackOverflow). It is interesting that although the values of the weights are stable within a site (StackOverflow or Twitter), the weights change dramatically when looking across the sites. This illustrates that the impact of a user's past history relative to the current context can be domain specific, which means that for best results the weights



*Figure 31.* Coefficient values for each component of the full models for StackOverflow. Models used are the best-performing Bayesian and random permutation models (i.e., Bayesian and RP have entropy weighting, RP converts context activation from correlation to log odds). Error bars represent the standard deviation of coefficient values for each component. The standard deviation is used instead of the confidence interval of the mean so that the plots show the overall stability of each coefficient across the different runs.

for each of the terms should be recalibrated in new scenarios. However, the underlying reason that the weights change between the sites cannot be answered from this study alone. This relative impact may be changing from either a true alteration in the underlying weight of the prior compared to context for different domains, or simply due to the fact that the amount of information collected for each of the terms changes between the domains (less information leads to lower weighting in general).

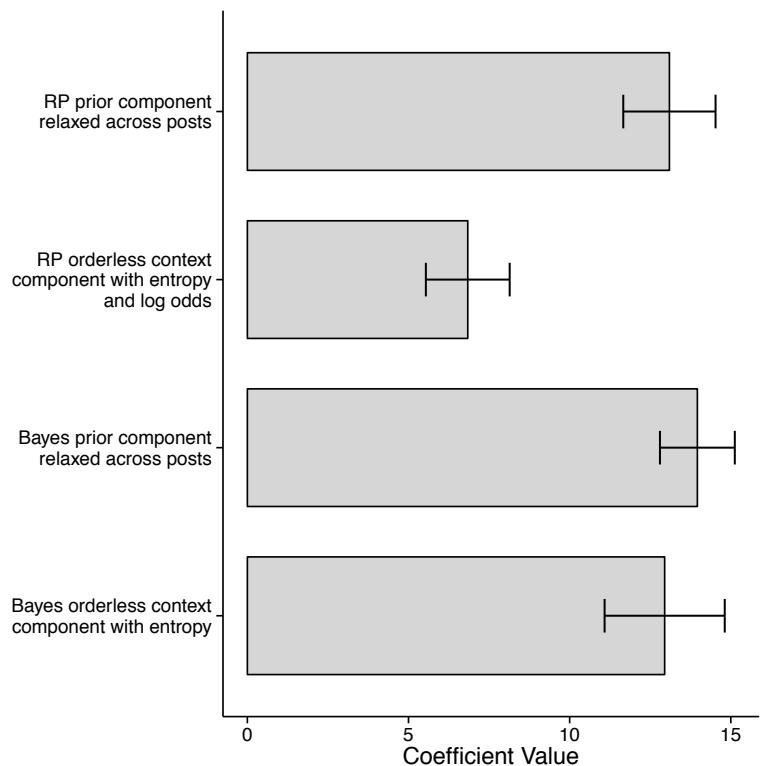


Figure 32. Coefficient values for each component of the full models for Twitter

## Chapter 6: Combining Predictors on the Popular-Users Dataset

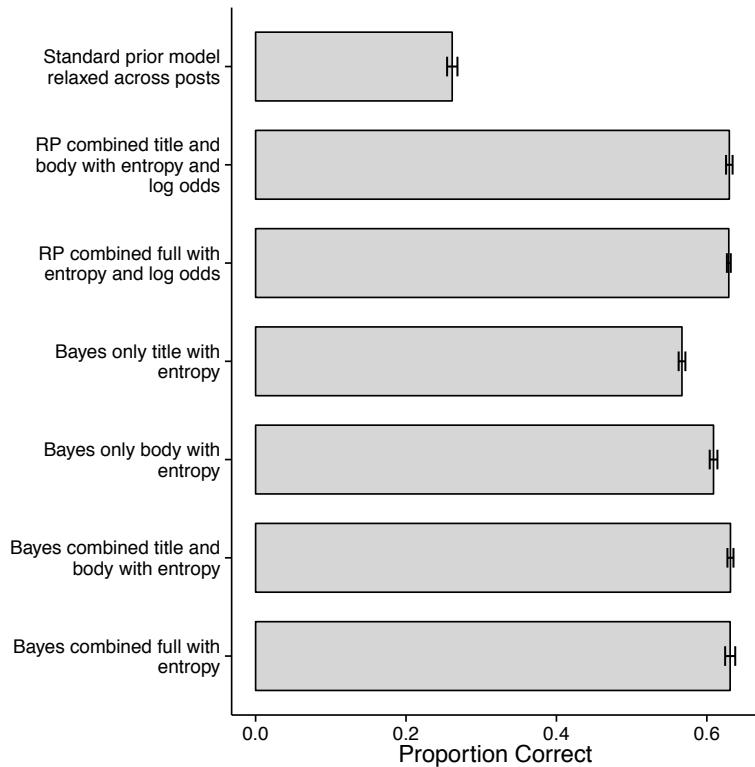
Chapter 4 isolated the prior component of the model and analyzed accuracy on the popular-users dataset. Chapter 5 added the context components to the models and analyzed accuracy on the Twitter popular-hashtags and StackOverflow randomly-sampled datasets. Chapter 6 shows the results when the combined prior and context models are tested across all of the datasets. Looking at model behavior across the datasets provides a way to show how the importance of the prior component relative to the context component can change depending on specific properties in the dataset that is studied. Also, an experiment to a user-custom context co-occurrence matrix was attempted on the popular-users dataset. This was tried because it could be theoretically interesting to see if model performance improves when the context co-occurrence matrix is customized to each user.

### Impact of Prior vs. Context

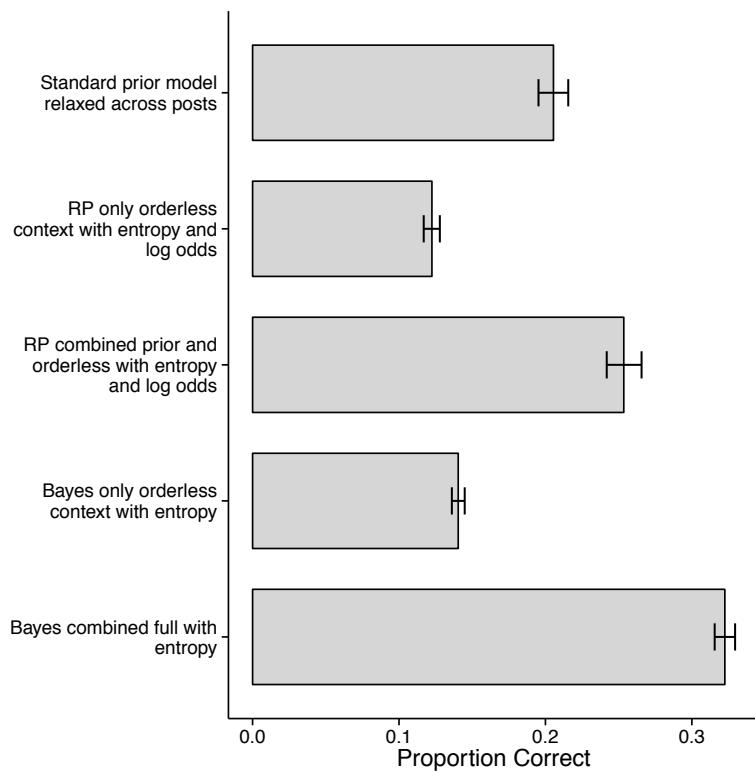
Adding a user's prior tag history as an additional component has been shown to improve performance, but is this always the case? If enough context information has been collected, maybe a context-only model can perform just as well as a combined context and prior model. If this were the case, then it may be less important that the random permutation model does not include a prior term by default.

**Method.** Full models were tested against context-only models for both sites. The four Twitter popular-hashtags datasets and the StackOverflow randomly-sampled dataset were used for the test.

**Results.** The results are depicted in Figures 33 and 34. For StackOverflow, accuracy remains the same when the prior component is removed for both the Bayesian (.63 to .63) and random permutation (.63 to .63) models. Accuracy does dramatically decrease for Twitter when the prior is removed, however (.32 to .14 for Bayesian, .25 to .12 for random permutation).



*Figure 33.* Model accuracy when prior is removed for StackOverflow randomly-sampled dataset



*Figure 34.* Model accuracy when prior is removed for Twitter popular-hashtags dataset

**Discussion.** These results show that prior information is not always needed when a large amount of context is available and not much information has been collected about each user's past history. The StackOverflow randomly-sampled dataset selects posts across the entire dataset, regardless of each user's popularity or tag history. Across the dataset, each user only asks around five questions on average, so there is not a large amount of past tag history available for the prior component to use, which is why this component is not a strong predictor in this case.

However, can a context-only model perform just as well as a combined context and prior model when users with the largest amount of StackOverflow history are tested? If each user's tagging history is a useful predictor for StackOverflow, then the prior component should at least generate unique predictive variance in the case where users with large amounts of history are tested.

**Method.** The same StackOverflow popular-users dataset used to test the prior component in isolation was used to test the prior component when context was added. However, since computing activations for the context component take an order of magnitude more wall-clock time than computing the prior component, not all observations could be included for the test. To reduce the observations, the most recent 50 posts from each user in the datasets were selected. The total number of users sampled was also reduced to 40 users for StackOverflow and 80 users for Twitter. Using the most recent observations for each user ensures that the observations tested have acquired the greatest amount of past tagging history in the datasets. So if a context-only model performs just as well as the full model in this case, it would be surprising.

**Results.** The results comparing the full model to a context-only only model for the StackOverflow popular-users dataset are depicted in Figure 35. In this case, past tag history does contain unique variance for accurately predicting tag use. Accuracy improves for both the Bayesian (.51 to .63) and random permutation (.47 to .58) models when the prior model component is combined with context compared to context alone.

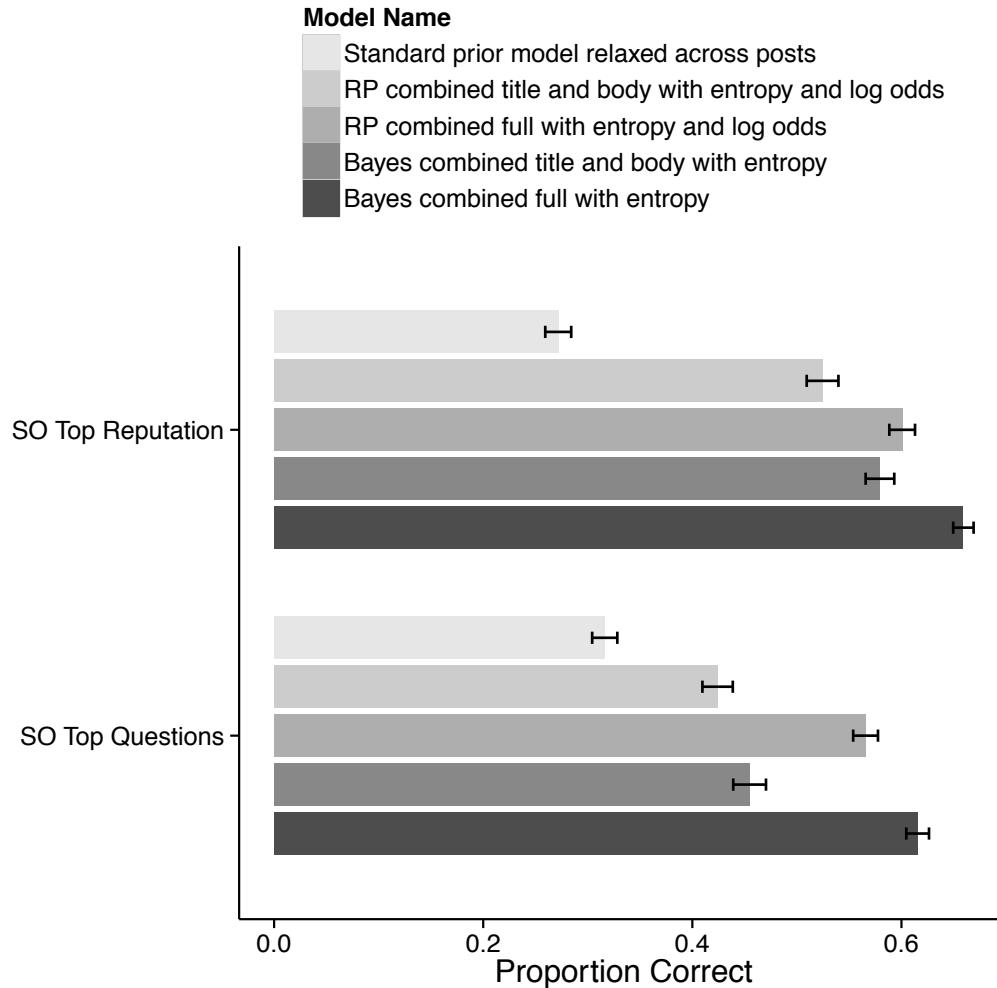


Figure 35. Model accuracy when prior is removed for StackOverflow popular-users dataset

**Discussion.** This shows that incorporating past tagging history is important even for a site like StackOverflow where a large amount of context information has been collected. However, for this type of site it only becomes important when a reasonably large amount of a user's past tagging history has been collected. For a randomly sampled user on StackOverflow, a context-only model captures the majority of the predictive variance.

This raises an important applied question: Exactly how much past user behavior is enough for a site like StackOverflow to benefit from adding past user behavior as an additional predictor? When the model predicts tags from a random sample of StackOverflow users, adding the prior component to the model does not significantly increase model accuracy compared to using context alone. However, when the model

predicts tags from a set of users that ask the most questions on the site, adding the prior does significantly increase accuracy. For the StackOverflow dataset, a user asks on average 5 questions and adds 3 tags to a post. So the model uses approximately 15 prior observations when computing the prior component for a randomly-sampled user on the site. For the popular-users dataset used to generate the results in Figure 35, a user asked 180 questions on average. So for this dataset, the model uses approximately 540 prior observations when computing the prior activation. So for the StackOverflow site, the prior component begins to generate unique predictive variance above and beyond the context component somewhere between when a user has generated 15 and 540 prior observations. This result will be different for other sites that have collected a different amount of contextual information. For sites like StackOverflow where the contextual component is already a strong predictor, it will take a larger number of prior observations before adding the prior component becomes beneficial. Alternatively for sites like Twitter where contextual information is much more noisy and not as strong of a predictor, it will take fewer observations before adding the prior becomes beneficial.

### User-customized Co-occurrence Matrix

The co-occurrence matrix for the context component of both models has been built by aggregating across users and collecting all instances of context words and associated tags. That is, the default co-occurrence matrix is not customized to each user, so the same matrix is used for all users. This is commonly done because there is usually not enough context by tag co-occurrences for a single user to generate a stable co-occurrence matrix. If there was enough information, accuracy would most likely improve since it is not likely that every user's word by tag co-occurrence profile is exactly the same. However, since several hundred posts for each user in the popular-users datasets are available, an analysis was tried where user-custom co-occurrence matrices were used.

**Method.** Separate co-occurrence matrices were built for each user in the sampled StackOverflow and Twitter popular-users dataset. The same sampling method used to test the impact of the prior component compared to the context component for the popular-users datasets was used, and the co-occurrence matrices were built from the oldest posts that were not included in the testing set. So for each user, the co-occurrence matrix was created from their oldest posts, up to (and not including) their 50 newer posts that were used to test model accuracy.

**Results.** Model accuracy when using a user-customized co-occurrence matrix for StackOverflow and Twitter is shown in Figures 36 and 37. The results compare accuracy for models that use a global co-occurrence and user-specific co-occurrence matrix. For StackOverflow, model accuracy for the user-specific co-occurrence matrix (labeled as “user sji”) can be compared against the default global co-occurrence matrix. Accuracy decreases dramatically when a user-specific co-occurrence matrix is used for both the Bayesian (.64 to .35) and random permutation (.58 to .36) models.

The results are similar for Twitter. However, the entire dataset is not available for Twitter, so a global co-occurrence matrix condition can not be tested, since the popular-users dataset samples across a wide range of hashtags, and due to Twitter’s API restrictions it would be infeasible to collect all posts that use each of the hashtags in the dataset. But user-specific co-occurrence matrices can be built, since the popular-users dataset does contain all context and hashtag observations for each user.

Model accuracy when only the single user-customized context predictor is used is relatively poor (.17 and .21 for Bayesian and random permutation), and the prior term is a much better predictor (.37). Further, when this component is combined with the prior component, model performance improves only slightly over performance when only the prior is used (.37 to .39 for both the Bayesian and random permutation).

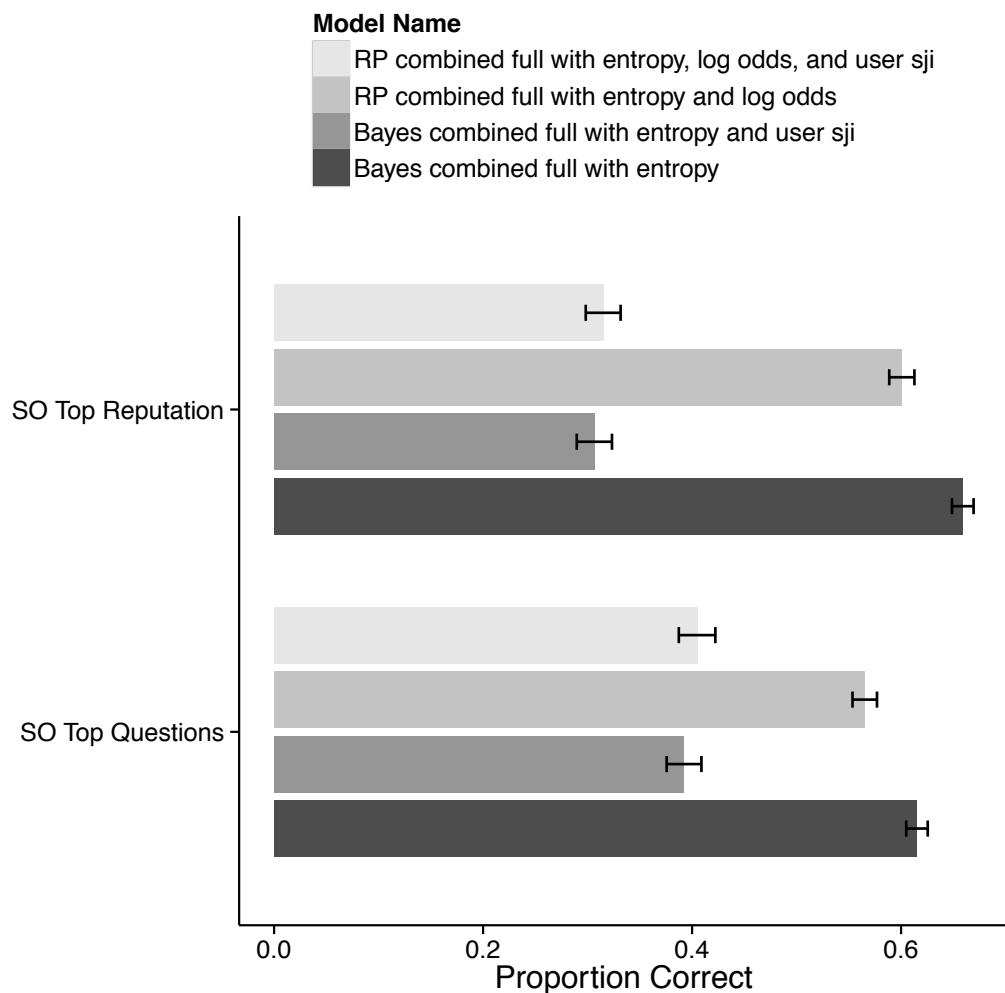


Figure 36. Model accuracy with user-customized context component for StackOverflow. A model name with “user sji” uses a co-occurrence matrix customized for each user instead of a single global co-occurrence matrix.

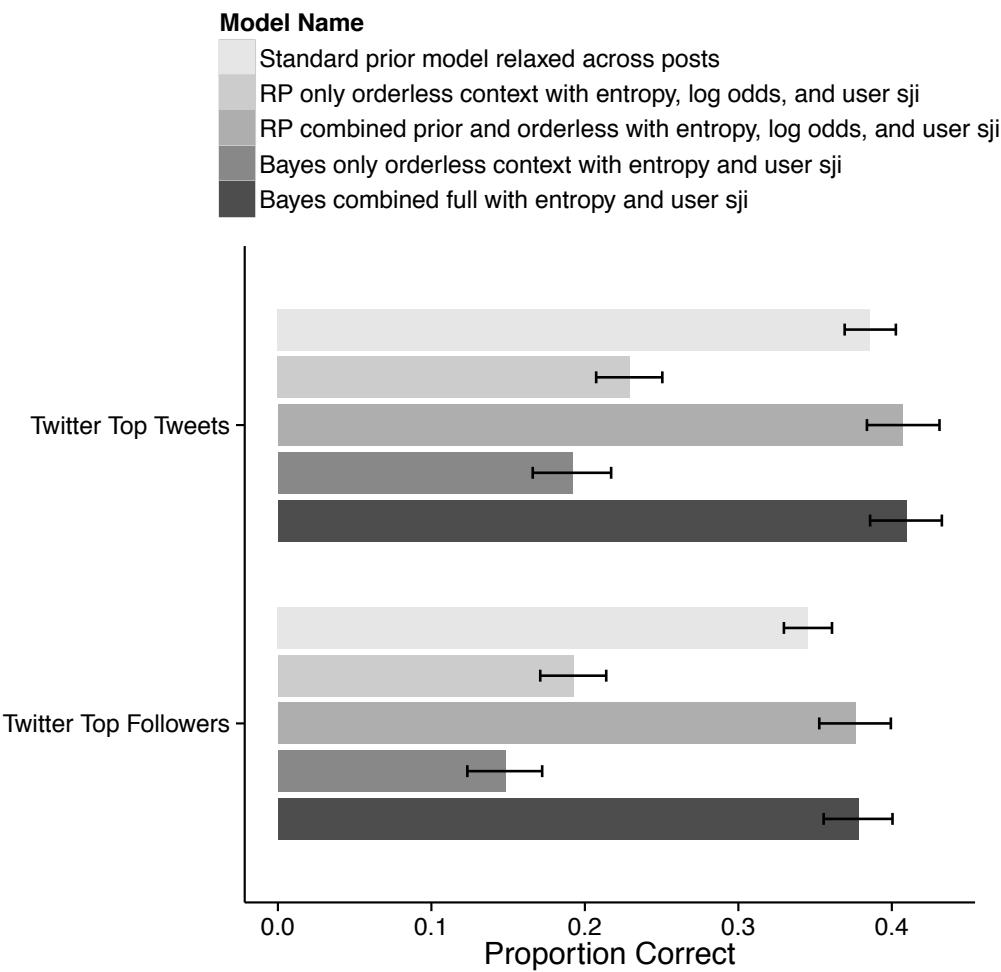


Figure 37. Model accuracy with user-customized context component for Twitter

**Discussion.** Both models require large numbers of context co-occurrence observations to make accurate predictions from context information. Otherwise, the co-occurrence matrices do not have enough data to properly stabilize. Millions of posts were used to generate the global co-occurrence matrices for Twitter and StackOverflow. If this number of posts had been collected for a single user, then the user-customized co-occurrence matrix for that user should lead to more accurate predictions compared to using the global matrix. However, in this study only around 100 posts were used to build the user-specific matrices for each user.

This is most likely why model accuracy was so poor when user-specific matrices were used: Even for the large datasets analyzed in this study, there is still not enough history collected for each user to build customized co-occurrence matrices. The prior model component can be customized for each user, since the prior is only concerned with tag use, and there are a reasonably small set of tags that the model is choosing from. However, the context component is built from the set of all of the prior tags crossed with all of the possible words, which is a much larger space than just the set of tags. So building a stable context component requires a much larger set of data than the prior component.

That said, if a user only chooses a small set of tags (so the tag space is small), and computing a global co-occurrence matrix is infeasible (e.g., with the Twitter popular-users dataset), then it may be worthwhile to add the user-custom context model component as an additional predictor. As was shown for Twitter, model accuracy can slightly improve over a model that just uses a custom prior, even when the co-occurrence matrix only contains around 100 posts.

Further, if the number of co-occurrences is small, and a user-custom co-occurrence matrix is used, it may be advantageous to use the compressed random permutation model instead of the uncompressed Bayesian model. For the Twitter popular-users dataset, only a user-custom co-occurrence matrix could be built, and when this matrix is used the compressed random permutation model performed better than the uncompressed Bayesian

model. When the global co-occurrence matrix was removed from the StackOverflow dataset, and a user-custom matrix was used instead, the accuracy for the random permutation model decreased less than the Bayesian model. This provides further evidence showing that there is an interaction between dataset size and optimal model: At lower numbers of co-occurrences the compressed random permutation model is able to perform better than the Bayesian model, and the Bayesian model surpasses the compressed model at higher co-occurrence counts once the cells in the full co-occurrence matrix stabilize.

## Chapter 7: General Discussion

The general discussion is separated into two parts. The first section will describe the overall results and reemphasize key findings. The final section will discuss general theoretical implications of the results and describe future applications and directions.

### Overall Results

Table 5 summarizes model accuracies for full models (all components) on all of the datasets tested in the study. After modifying the models by adding an entropy weighting mechanism to both models and a prior term to the random permutation model, accuracy for the compressed random permutation model was nearly equal to the Bayesian model. The largest difference between the two models was on the Twitter popular-hashtags dataset (7 percent). Both models performed nearly equal for the StackOverflow datasets tested, and both models were much more accurate when tested on StackOverflow compared to Twitter.

Table 5

*Summary of accuracy for each model and dataset. The Bayesian model includes entropy weighting for the context words, the RP model includes entropy weighting and log odds transformation when combining the context and prior components. Both models use a user-customized co-occurrence matrix for the Twitter popular-users dataset, since a global co-occurrence matrix is not available for this dataset.*

Site	Dataset	Model	Accuracy
StackOverflow	popular-users	Bayesian with entropy	.63
		RP with entropy and log odds	.58
	randomly-sampled	Bayesian with entropy	.63
		RP with entropy and log odds	.63
Twitter	popular-users	Bayesian with entropy and user sji	.39
		RP with entropy, log odds, and user sji	.39
	popular-hashtags	Bayesian with entropy	.32
		RP with entropy and log odds	.25

Overall, both the random permutation and Bayesian declarative memory models fit the observed tagging behavior in StackOverflow and Twitter relatively well. The fits for StackOverflow in particular are quite high, especially given the large tag space on the site.

These results provide support for the idea that choosing a hashtag when composing a tweet and tagging a StackOverflow post is akin to a declarative memory retrieval process.

However, both models were not 100 percent accurate, and the amount of variance left to predict was much larger than what would be observed if retrieval noise was included in the model. So it is most likely the case that the task is not done solely with declarative memory processes.

Alternatively, perhaps authors of posts are taking advantage of context information that is not currently represented in the model. For example, time of day might be a useful predictor of tag use on Twitter. Preotiuc-Pietro and Cohn (2012) built a tag-prediction model for Twitter that was based primarily on temporal cycles of hashtag use. This technique could be naturally incorporated into a co-occurrence framework by including time of day as an additional context by tag co-occurrence for each post. This would allow time of day information to be used alongside the rest of the context information in the words of the post. A similar technique could be used to add additional context predictors to the model. These co-occurrences could either be added on as additional columns to the existing co-occurrence matrix, or a separate co-occurrence matrix could be used for each additional context component, similar to how separate matrices and context predictors were created for the words in the title and body of a post. Including additional context predictors such as time of day may help improve model accuracy, and perhaps it is the case that the rest of the variance left to predict is from these context components that the author is using but are not yet included in the model.

Anderson and Schooler (1991) also looked at similar tagging tasks with large datasets when deriving the prior component of the declarative memory theory. Anderson and Schooler examined, for example, the frequency of word choice over time used for the title of articles for the New York Times, and used the results to derive the power law equation for the prior term in the declarative memory model. ACT-R's declarative memory theory claims that the system has evolved to make predictions that closely match the statistical

structure of the environment. Using a power-law equation to model future tag use based on past tag use is a reasonable statistical approach to the problem. It is also the case that this same equation is an accurate model for the prior component of declarative memory, as was shown by Anderson and Schooler in the original derivation, and as is supported in the results from this study.

**Model Accuracy Differences Across Sites.** Model accuracy for StackOverflow was much higher overall than for Twitter. The Twitter popular-hashtags dataset contained 400 unique hashtags for each of the four dataset subsets. The StackOverflow randomly-sampled dataset contained 34,377 unique tags, since it sampled posts across the entire site. As the model is tasked to predict fewer and fewer hashtags, it should become more accurate (less chance of making an incorrect prediction). However, even with a much smaller tag set to predict, model accuracy for Twitter was still not nearly as high as StackOverflow. This is likely due to the fact that tag frequency use drops off sharply for both StackOverflow and Twitter. This means that the models on both sites are generally switching between the top few hundred tags when making predictions. So, increasing the total unique number of tags past that point makes only a small negative impact on model accuracy.

However, there was a large difference in total number of users for StackOverflow and Twitter. A user on StackOverflow contributed 2.5 times more posts to the StackOverflow randomly-sampled dataset than the Twitter popular-hashtags dataset. Also note that the large co-occurrence matrices for StackOverflow and Twitter that contained millions of documents were not user specific. This means that the co-occurrence matrix for Twitter was built from a much larger set of unique users compared to StackOverflow. It is likely that co-occurrence matrices for individual users are correlated but not equivalent. So, as the co-occurrence matrix is built from a larger number of unique users, a greater amount of cross-user noise is introduced into the matrix. This noise would decrease model accuracy since the contextual component of the model would lose predictive information by having a

noisier co-occurrence matrix. This is most likely one of the main reasons that model accuracy for Twitter is much lower than StackOverflow: Tag prediction becomes much more challenging as the number of users on a site increases.

The second most-likely reason that model accuracy is much higher for StackOverflow is that the co-occurrence matrix for this site contains much more data due to the large document size for the site. The co-occurrence matrix for the StackOverflow randomly-sampled dataset contained 20 times more observations than the matrix for one of the Twitter popular-hashtags datasets. Model accuracy generally increases as more observations are used to generate the co-occurrence matrices, since this allows each cell in the matrix to stabilize to its true value.

**Representation.** Given that the storage space required to represent the co-occurrences for the random permutation model is dramatically smaller than the uncompressed Bayesian model, and that the compression is lossy and essentially random (i.e., no SVD), it is quite interesting that the random permutation model performs nearly as well as the full Bayesian model for this task.

These results pose an interesting and important fundamental question about declarative memory: How, precisely, is the neural system configured to represent co-occurrences for declarative memory? Since at the behavioral level both models perform similarly, it is possible that the neural system is using either representation to store co-occurrences. Vector-based models are simple enough that they can easily be implemented in a neural network. Perhaps a form of vector-based compressed storage is being used at the neural level, since it can be implemented in neurons and is an efficient way for the machinery to generate predictions that closely match the statistical structure of the environment (i.e., that closely match the uncompressed Bayesian representation).

As neural imaging systems improve, this question of neural representation can be best answered by looking directly at the neural activity. However, there are possible behavioral studies that could be done to provide evidence that may favor one representation over the

other: The largest differences between the Bayesian and vector-based models happen with small numbers of co-occurrence observations. The vector-based model begins to generate more accurate predictions sooner than the Bayesian models, since the noise introduced from the compression helps smooth out activation values until a larger number of co-occurrences are observed and the co-occurrence values stabilize. For a behavioral study, novel words could be introduced alongside a corpus of known words, and the participant could be asked to make retrievals of the novel words and known words as more novel words are introduced. Both models would make different predictions about the rate at which activation values for the novel words stabilize relative to the known words, and such make different predictions about retrieval error rates over time. Learning a new language might be a domain where large amounts of these sorts of retrievals could be collected. For instance, the online site Duolingo might contain enough collective data about language learning and word translation to better tease apart the two context models.

**Word Order.** Also, the relatively strong performance of the random permutation model is not due to the model's ability to represent word order. Performance is similar for the random permutation and Bayesian models when the co-occurrence matrix for both models is the same, and not based on word order. Performance also does not improve by much when word order is added as the last predictor in the random permutation model. The random permutation model seems to work so well primarily due to the way that the full co-occurrence representation is compressed.

**Compression.** As previously noted, compression for the random permutation model is essentially random. That is, no SVD is done to identify the most predictive components, so the compressed representation used by the random permutation model does not contain nearly as much of the variance from the original co-occurrence matrix that it could have. However, performing a full SVD on a co-occurrence matrix as large as the ones generated for StackOverflow and Twitter is computationally infeasible. Also, when deriving the LSA approach, Landauer and Dumais (1997) explicitly state that they make no claims that the

brain is computing an SVD at the neural level; only that the neural machinery is set up such that the representation makes predictions that are similar to what would have been generated if the matrix decomposition for an SVD would have been computed. Perhaps the randomly-compressed representation used for the random permutation model is the way in which the brain tries to efficiently achieve the full LSA- and SVD-like approach that Landauer and Dumais propose.

**Corpus Size.** The random permutation model was more accurate than the Bayesian model at smaller corpus sizes, while the results were flipped for larger document sizes. This is interesting from a practical standpoint because the corpus size is rarely chosen in declarative memory research, and is constrained primarily by the amount of data available in the domain being studied. If a researcher is working within a domain that has a relatively small corpus size (generally near or less than 1000 documents), it may be worthwhile to test the random permutation model in this domain, as this compressed model may be more accurate than the uncompressed Bayesian model.

Further, it is interesting to study the effect of model accuracy on corpus size because people require exposure to only a small number of documents before tagging documents appropriately. A user asked around five questions on average for StackOverflow, model accuracy was high for StackOverflow, but the model required training on 1 million documents before achieving that accuracy. The user only needed to “train” on five questions and the model needed to train on 1 million questions.

This is a large discrepancy, but this does not necessarily invalidate the declarative memory models used for this research. First, the random permutation model was still accurate at smaller document sizes, so perhaps this result provides evidence to favor the random permutation model over the Bayesian model. However, the smallest corpus size tested for these models was 1000 documents, which is much greater than the average number of questions or tweets that a user contributes to the site.

Second, the models may require large amounts of training because the co-occurrence

matrices are being built from scratch. That is, the models are being trained on the StackOverflow and Twitter domains without any prior general co-occurrence information to start with. Presumably, when a person starts using a site, they can leverage all of their past exposure to different word co-occurrences and then simply add on top of that base the domain-specific co-occurrences. In that since, one could argue that the models are undertrained on generic co-occurrences (people are exposed to many more than one million documents in their lifetime) and overtrained on specific co-occurrences. It may be interesting to test these memory models on a hybrid co-occurrence matrix, where the majority of the observations are taken from a generic domain (e.g., american literature) and a minority are domain-specific (e.g., StackOverflow word, tag pairs).

**Attentional Weight.** The Bayesian declarative memory model for ACT-R has an attentional weight term  $W_j$  that can be used to attenuate or remove stop words. The random permutation model was also easily modified to have an attentional weight term by attenuating each word's set of ones and negative ones values by that word's attentional weight. Both a method for stop-word removal (frequency filtering) and stop-word attenuating (entropy weighting) performed well for both models. Also, using a proper method for handling the low-predictor words produced some of the largest improvements in model accuracy compared to the other explorations (e.g., compared to word order, properly adding terms, compression amount for random permutation representation). Since the ACT-R declarative memory theory already has a component for attentional weight, it may be the case that the neural system performs some form of stop-word weighting when computing activations, and this is reflected in the attentional weight term. That is,  $W_j$  can be defined mathematically, has the same general form across tasks, and both the frequency-filtering and entropy-weighting methods for stop-word removal are likely first candidates for the definition. This claim, however, would require that no top-down activation is used to compute the attentional weight, which would mean that the  $W_j$  is not under conscious control, which is not likely to be correct. Perhaps it is a combination of

top-down and bottom-up systems that determine  $W_j$ , and the entropy-weighting and frequency-filtering techniques are methods of formalizing the bottom-up component.

## Implications and Future Directions

**Generalizing Results.** It seems likely that the entropy method for computing the attentional weight term  $W_j$  may be applicable to other declarative memory tasks. The entropy method is parameter free (unlike the frequency-filtering method), completely data driven (much like the co-occurrence matrix), and does not require tuning. It is common to handle stop words in some manner when working with NLP and large text documents. So, if these large natural language documents are used to test declarative memory, then the entropy method may be a more cognitively plausible way of dealing with stop words rather than filtering based on a predetermined list or based on the frequency count in the dataset, particularly since the entropy method substantially increases model accuracy for these datasets and since it is parameter free.

Adding a prior component to the random permutation model also increased model accuracy for both StackOverflow and Twitter. It is unsurprising that this occurred. However, it is quite surprising that this is the first time that the prior component has been included and tested in a random permutation model. If the random permutation model is explored as a declarative memory theory for other datasets, it should be worthwhile to include the prior component for those explorations. Although it did not seem to matter much how the prior component is combined with the context component in this study, there may be better ways than the log-odds transformation to combine the terms. Further research could focus on exploring a larger set of methods for combining the context component of the random permutation model with the prior component from the ACT-R theory.

**Activation Calculation.** The Bayesian and vector-based memory models differ in how the co-occurrence matrix is represented (lossy compression for vector-based models).

They also differ in how the activation of each tag is calculated, given the different representations. The vector-based memory models, like LSA, compute a correlation between the context and each tag, while the Bayesian models sum the log likelihoods that each word in the context occurs with each tag. These two methods for computing the activation are correlated but not identical. Further research could try to tease apart how model accuracy is independently influenced by the representation (compressed or not) and the activation calculation (correlation or log likelihood). To do this, four models could be constructed (uncompressed and compressed representation crossed with correlation and log likelihood activation) compared to the two in this study, so that the effects of representation and activation calculation could be partialled out. This would allow, for example, the method of compression for the random permutation vector-based model to be analyzed independent of activation calculation. The results would help determine the specific representation that the declarative memory system uses to represent co-occurrences, and also the specific activation calculation that is carried out by the system.

**Future Applications.** The tag-recommendation models developed through this research should be relevant to real-world tasks. Model accuracy is high enough (particularly for StackOverflow) for the developed models to be used as a foundation for tag recommendation systems that are deployed on Twitter or StackOverflow. However, the focus for this research was to evaluate the accuracy of two cognitively-plausible declarative memory models, so the models were more constrained than general machine learning models of tagging. So it is unlikely that these general models of declarative memory will outperform specific machine learning models that have been optimized for a specific tagging task. Nevertheless, one of the advantages of the general declarative memory models is that they are task agnostic, so it would take minimal time to deploy these models on a new tagging task.

**Summary.** The end result of this research is [1] a better understanding of how a user's past tagging history influences future tag use, [2] a more thorough evaluation of the

strengths, weaknesses, and differences between two cognitively-plausible memory models on large-scale real-world retrieval tasks, and [3] two efficient implementations of the models for making user-customized tag predictions for StackOverflow posts and Twitter tweets. This work has shown how past user history is important, and how a context-only model like the random permutation model can be modified to include the prior component. The work has also shown how the strength of the random permutation model may not be in its ability to represent word order, and rather in its ability to efficiently compress the full co-occurrence representation used for the Bayesian model. Also from a methodological perspective, this work has shown how large collections of publicly-available behavioral datasets can be used to explore and test different theories of declarative memory and their individual architectural components. Several different formalisms for ACT-R’s attentional weighting term were also explored, and an entropy-weighting technique was found to perform well without requiring any parameter tuning. Finally, this work has helped raise the question about how co-occurrence observations are represented at the neural level, since two models with quite different representations performed similarly well. The results from the study and the new research questions raised will help to better understand how imposing specific architectural constraints on memory models influence what information is retrieved.

## References

- Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* New York, NY: Oxford University Press.
- Anderson, J. R. & Milson, R. (1989). Human memory: An adaptive perspective. *Psychological Review*, 96(4), 703.
- Anderson, J. R. & Schooler, L. J. (1991). Reflections of the environment in memory. *Psychological Science*, 2(6), 396–408.
- Anderson, J. R. (1974). Retrieval of propositional information from long-term memory. *Cognitive Psychology*, 6(4), 451–474.
- Bennett, J. & Lanning, S. (2007). The Netflix prize. In *Proceedings of KDD Cup and Workshop* (p. 35). San Jose, CA.
- Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python*. O'Reilly Media, Incorporated.
- Budiu, R., Royer, C., & Pirolli, P. (2007). Modeling information scent: A comparison of LSA, PMI and GLSA similarity measures on common tests and corpora. In *Proceedings of the 8th Annual Conference of the Recherche d'Information Assistée par Ordinateur* (pp. 314–332). Pittsburgh, PA: Centre des Hautes Études Internationales d'Informatique Documentaire.
- Chang, H.-C. C. (2010). A new perspective on Twitter hashtag use: Diffusion of innovation theory. *Proceedings of the American Society for Information Science and Technology*, 47(1), 1–4.
- Diakopoulos, N. A. & Shamma, D. A. (2010). Characterizing debate performance via aggregated twitter sentiment. In *Proceedings of the 28th International Conference on Human factors in Computing Systems* (pp. 1195–1198). New York, NY: ACM.
- Douglass, S. A. & Myers, C. W. (2010). Concurrent knowledge activation calculation in large declarative memories. In *Proceedings of the 10th International Conference on Cognitive Modeling* (pp. 55–60). Philadelphia, PA: Drexel University.
- Dowle, M., Short, T., Lianoglou, S., with contributions from R Saporta, A. S., & Antonyan, E. (2014). *data.table: Extension of data.frame*.
- Dumais, S. T. (1991). Improving the retrieval of information from external sources. *Behavior Research Methods*, 23(2), 229–236.
- Efron, M. (2010). Hashtag retrieval in a microblogging environment. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 787–788). New York: ACM.

- Efron, M. & Golovchinsky, G. (2011). Estimation methods for ranking recent information. In *Proceedings of the 34th international ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 495–504). New York: ACM.
- Farahat, A., Pirolli, P., & Markova, P. (2004). *Incremental methods for computing word pair similarity* (tech. rep. No. TR-04-6-2004). PARC, Inc, Palo Alto, CA.
- Fu, W. T. & Pirolli, P. (2007). SNIF-ACT: A cognitive model of user navigation on the World Wide Web. *Human–Computer Interaction*, 22(4), 355–412.
- Godin, F., Slavkovikj, V., De Neve, W., Schrauwen, B., & De Walle, R. (2013). Using topic models for Twitter hashtag recommendation. In *Proceedings of the 22nd International Conference on World Wide Web* (pp. 593–596). International World Wide Web Conferences Steering Committee.
- Google. (2013). Google I/O 2013: Keynote. Retrieved from [http://www.youtube.com/watch?v=9pmPa\\_KxsAM#t=1h28m18s](http://www.youtube.com/watch?v=9pmPa_KxsAM#t=1h28m18s)
- Hannon, J., Bennett, M., & Smyth, B. (2010). Recommending Twitter users to follow using content and collaborative filtering approaches. In *Proceedings of the Fourth ACM Conference on Recommender Systems* (pp. 199–206). New York: ACM.
- Jansen, B. J., Booth, D. L., & Spink, A. (2008). Determining the informational, navigational, and transactional intent of Web queries. *Information Processing & Management*, 44(3), 1251–1266.
- Jansen, B. J., Liu, Z., Weaver, C., Campbell, G., & Gregg, M. (2011). Real time search on the web: Queries, topics, and economic value. *Information Processing & Management*, 47(4), 491–506.
- Jones, M. N. & Mewhort, D. J. K. (2007). Representing word meaning and order information in a composite holographic lexicon. *Psychological Review*, 114(1), 1–37.
- Kanerva, P., Kristofersson, J., & Holst, A. (2000). Random indexing of text samples for latent semantic analysis. In *Proceedings of the 22nd Annual Conference of the Cognitive Science Society* (p. 1036). Mahwah, NJ: Erlbaum.
- Kuo, D. (2011). *On word prediction methods* (tech. rep. No. UCB/EECS-2011-147). EECS Department, University of California, Berkeley.
- Kwak, H., Lee, C., Park, H., & Moon, S. (2010). What is Twitter, a social network or a news media? Categories and subject descriptors. In *Proceedings of the 19th International Conference on World Wide Web* (pp. 591–600). New York: ACM.
- Kywe, S. M., Hoang, T.-A., Lim, E.-P., & Zhu, F. (2012). On recommending hashtags in twitter networks. In *Social Informatics* (pp. 337–350). New York: Springer.

- Landauer, T. K. & Dumais, S. T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review, 104*(2), 211.
- Lee, U., Liu, Z., & Cho, J. (2005). Automatic identification of user goals in web search. In *Proceedings of the 14th International Conference on World Wide Web* (pp. 391–400). New York: ACM.
- Lewis, R. L., Vasishth, S., & Van Dyke, J. A. (2006). Computational principles of working memory in sentence comprehension. *Trends in cognitive sciences, 10*(10), 447–454.
- Li, T., Wu, Y., & Zhang, Y. (2011). Twitter hash tag prediction algorithm. In *The 2011 International Conference on Internet Computing*. Las Vegas: Citeseer.
- Mazzia, A. & Juett, J. (2009). *Suggesting hashtags on Twitter*. EECS 545 (Machine Learning) Course Project Report. Department of Computer Science and Engineering, University of Michigan.
- McCandless, M., Hatcher, E., & Gospodnetic, O. (2010). *Lucene in action, second edition: Covers Apache Lucene 3.0*. Greenwich, CT: Manning Publications Co.
- Owoputi, O., O'Connor, B., Dyer, C., Gimpel, K., Schneider, N., & Smith, N. A. (2013). Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of NAACL-HLT* (pp. 380–390). Atlanta: Association for Computational Linguistics.
- Pazzani, M. J. & Billsus, D. (2007). Content-based recommendation systems. In *The Adaptive Web* (pp. 325–341). New York: Springer.
- Pirolli, P. & Fu, W.-T. (2003). SNIF-ACT: A model of information foraging on the World Wide Web. In *User Modeling 2003* (pp. 45–54). Johnstown, PA: Springer.
- Plate, T. A. (1995). Holographic reduced representations. *IEEE Transactions on Neural Networks, 6*(3), 623–641.
- Preotiuc-Pietro, D. & Cohn, T. (2012). A temporal model of text periodicities using Gaussian Processes. *Proceedings of EMNLP*, 977–988.
- R Core Team. (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria.
- Recchia, G., Jones, M., Sahlgren, M., & Kanerva, P. (2010). Encoding sequential information in vector space models of semantics: Comparing holographic reduced representation and random permutation. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society* (pp. 865–870). Austin, TX: Cognitive Science Society.

- Rose, D. E. & Levinson, D. (2004). Understanding user goals in web search. In *Proceedings of the 13th International Conference on World Wide Web* (pp. 13–19). New York: ACM.
- Rutledge-Taylor, M. F. & West, R. L. (2007). MALTA: Enhancing ACT-R with a holographic persistent knowledge store. In *Proceedings of the 24th Annual Conference of the Cognitive Science Society* (pp. 1433–1439). Austin, TX: Cognitive Science Society.
- Rutledge-Taylor, M. F. & West, R. L. (2008). Modeling The fan effect using dynamically structured holographic memory. In *Proceedings of the 30th Annual Conference of the Cognitive Science Society* (pp. 385–390). Austin, TX: Cognitive Science Society.
- Sahlgren, M., Holst, A., & Kanerva, P. (2008). Permutations as a means to encode order in word space. In *Proceedings of the 30th Annual Conference of the Cognitive Science Society* (pp. 1300–1305). Austin, TX: Cognitive Science Society.
- Salton, G. & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5), 513–523.
- StackExchange. (2013a). Tag suggestion system. Retrieved from <http://tex.stackexchange.com>
- StackExchange. (2013b). Tag suggestion system. Retrieved from [www.superuser.com](http://www.superuser.com)
- StackExchange. (2014). Creative commons data dump. Retrieved from <http://data.stackexchange.com/about>
- Stanley, C. (2014). Repository for Dissertation. GitHub. Retrieved from <https://github.com/claytontstanley/dissertation>
- Stanley, C. & Byrne, M. D. (2013). Predicting tags for StackOverflow posts. In *The 12th International Conference on Cognitive Modeling* (pp. 414–419). Ottawa: Carleton University.
- Tsur, O. & Rappoport, A. (2012). What's in a hashtag?: Content based prediction of the spread of ideas in microblogging communities. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining* (pp. 643–652). New York: ACM.
- Turney, P. (2001). Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. In *Proceedings of the Twelfth European Conference on Machine Learning* (pp. 491–502). Berlin: Springer-Verlag.
- Twitter. (2012). Twitter hits half a billion tweets a day. Retrieved from [http://news.cnet.com/8301-1023\\_3-57541566-93/report-twitter-hits-half-a-billion-tweets-a-day](http://news.cnet.com/8301-1023_3-57541566-93/report-twitter-hits-half-a-billion-tweets-a-day)

Wickham, H. (2009). *ggplot2: elegant graphics for data analysis*. New York: Springer.

Zangerle, E., Gassler, W., & Specht, G. (2011). Recommending #-tags in Twitter. In *Proceedings of the Workshop on Semantic Adaptive Social Web* (pp. 67–78).

## Appendix 1: Support Code

All software code created for this research is publicly available on GitHub (Stanley, 2014).

**Directory Structure.** The repository is separated into three main folders: dissProject, firstSOProject, and writeups. The firstSOProject folder contains the support code for the Stanley and Byrne (2013) research. The dissProject folder contains support code for this dissertation research. The writeups folder contains all writeup projects created during the dissertation process (e.g., the LaTeX for this dissertation, Keynote presentations).

Within the dissProject folder, there are two main files that contain the majority of the code for the dissertation research: scrapeUsers.py and preProcess.R. The python file contains the necessary code to scrape the Twitter site for tweets, import results into the Postgres database, and perform various cleanup functions that were easiest to do in Python. The preProcess.R file contains all modeling and analysis code for the project. Within the file, there are implementations for the Bayesian and random permutation models. Also there is code to test the models by running each model through the various StackOverflow and Twitter test datasets.

**Model Implementations.** The Bayesian model is implemented using the data.table package, so that fast joins can be performed on the table when computing the strength of association values. The random permutation model is represented as a matrix, so that correlations between the model and context vectors are fast. The matrix is not sparse, since there are currently no sparse matrix packages for R that have a vectorized function to compute correlations between a vector and a matrix.

The Bayesian model requires slightly more time to compute activation values and also more memory space to store the representation. This is because the random permutation model is compressed and only needs to store around 2048 rows in the co-occurrence matrix. However, both models are implemented efficiently enough so that all

analysis runs and activation computations can be kept in memory. 64 GB of memory was required to perform the analysis, particularly on the largest Twitter (3 million) and StackOverflow (1 million) corpus sizes.

**Design Decisions.** All model manipulations (e.g., using an entropy weighting metric, using a model with only the prior term) were expressed as part of a configuration object. That object is a list of configuration name, value pairs. The configuration object is passed to the analysis code, which determines the specific type of run that is performed.

This configuration design enables a large amount of code reuse and overlap. This helps minimize the likelihood of a programming mistake, since functions are reused multiple times and overall static code length is kept to a minimum.

**Testing.** Each analysis run produced a set of result files. Those result files were checked into the repository so that changes could be tracked. Code changes were tested by looking at differences between previously checked in output files and the output files produced after a code change was made.

A small amount of unit testing was also performed. The core behavior of each model implementation (Bayesian and random permutation) was tested in isolation to ensure that the activation values returned by the models were mathematically correct. Unit test code is located within dissProject/test.

**Parallelization.** None of the analysis code is finely parallelized. Most of the inner loop in R ends up running in compiled C, and execution within those function calls occurs on a single core. However, since multiple models were explored, each different model configuration was executed in parallel. This type of course-grained parallelization was achieved using R's built-in Parralel package. Also some parallelization was done at the OS level when various make targets would run in parallel.

**Database.** All raw tweet and post data were stored in Postgres tables. Also, after the documents were tokenized, the tokenized versions of the raw data were stored in Postgres tables. When R ran an analysis, it would query the tables to retrieve the correct

set of tokenized words and tags. So there is a lot of R code that uses the RPostgreSQL package to communicate with Postgres as the analysis is running.

**Running the Analysis.** All analysis runs were scripted and can be run by using the appropriate target in dissProject/Makefile. The three main targets to run the models through the prior-only runs on the popular users dataset, the full models on the randomly-sampled and popular-hashtags datasets, and the full models on the popular users dataset are “parrunPriors”, “parrunContext”, and “parrunPUsers” respectively. Each of these targets takes tens of hours to complete. “parrunContext” is the longest and takes around 72 hours on a desktop machine with 64 GB of memory and 24 virtual cores.

Smaller test runs were also created so that model changes could be more rapidly developed. These typically use smaller Twitter and StackOverflow corpus sizes (e.g., only 10,000 StackOverflow posts) to create the co-occurrence matrix and a smaller number of posts to test the models on (e.g., 20 instead of 500). Running these functions takes around a minute to complete. To run these, the R function is called directly (i.e., not a make target) within an already-running R session. As an example, the “runContext20g1s6” function takes only a few minutes to complete, runs the models on a corpus size of 1000, with only a single run per model, and only 20 sampled posts for each run.