EVALUATION OF TOPIC MODELS FOR CONTENT-BASED POPULARITY PREDICTION ON SOCIAL MICROBLOGS

by

Axel Magnuson

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Boise State University

May 2016

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Axel Magnuson

Thesis Title: Evaluation of Topic Models for Content-Based Popularity Prediction on Social Microblogs

Date of Final Oral Examination: 17 December 2015

The following individuals read and discussed the thesis submitted by student Axel Magnuson, and they evaluated his presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

María Soledad Pera, Ph.D.	Chair, Supervisory Committee
Timothy Andersen, Ph.D.	Member, Supervisory Committee
Edoardo Serra, Ph.D.	Member, Supervisory Committee

The final reading approval of the thesis was granted by María Soledad Pera, Ph.D., Chair of the Supervisory Committee. The thesis was approved for the Graduate College by John R. Pelton, Ph.D., Dean of the Graduate College.

Dedicated to my many teachers

ACKNOWLEDGMENTS

The author wishes to deeply thank Dr. María Soledad Pera and Dr. Vijay Dialani for their invaluable guidance in my degree. Special recognition is due to Dr. María Soledad Pera, whose exceptional effort in guiding this thesis went far above the call of duty. He would additionally like to thank the wonderful staff and faculty of Boise State University for their persistence and diligence in educating him, alongside his many colleagues. Among them, he wishes to name Dr. Amit Jain, Dr. Timothy Andersen, and Dr. Elena Sherman for their exceptional capacities as educators.

Furthermore, the author owes any of his achievement in life to the kind generosity of the those who have guided and supported him throughout his life. In particular, he would like to thank his Mother, Father, and Brother for their profound support and love. He would also like to thank a long line of inspiring teachers including David Pover, Tatia Totorica, AlejAndro Anastasio, and Dr. Arun Ram. The author wishes to acknowledge the incredible impact that all of those above have had on his life and their utter necessity in the production of this work. He owes thanks to the kind support of his many colleagues, especially Deepa Mallela, Nevena Dragovic, and Jim Pelton. Finally, he would like to thank Dr. Vijay Dialani and Dr. Edoardo Serra for their roles in providing the funding necessary to support the author throughout his endeavor.

ABSTRACT

Online social networks are an increasingly central medium of communication in the 21st century. We have seen a proliferation of competing social networks that differentiate themselves by serving different niches of communication. Among these, Twitter has risen to prominence as a leader among microblogging communities, characterized by publicly visible 140-character messages called tweets. The wide visibility of Twitter messages has enabled some users to curate large followings, and has facilitated content creators who wish to reach as many viewers as possible. Researchers have since investigated many methods for predicting which messages will become popular or even go viral on Twitter. Although there are many facets to this research problem, and various methods of approaching it have been proposed, we note that anyone who wants to create a popular Twitter account will sooner or later have to produce popular content. In this study, we investigated the content-based approach of predicting popular tweets based only on the text they contain. Particularly, we asked whether topic models can be used to identify topics of discussion that are more likely to be associated with popular tweets. In the process, we explored methods for collecting and processing a large-scale corpus of Twitter content. Our experiments found that while topic-based prediction methods could lead to effective popularity prediction, they were outperformed by other, simpler content-based methods.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AUC – ROC Area Under Curve

FPR – False Positive Rate

LDA – Latent Dirichlet Allocation

LLDA – Labeled Latent Dirichlet Allocation

OANC – Open American National Corpus

OSN – Online Social Network

PGM – Probabilistic Graphical Model

ROC – Receiver Operating Characteristic

sLDA – Supervised Latent Dirichlet Allocation

TFIDF – Term Frequency Inverse Document Frequency

TPR – True Positive Rate

UGC – User Generated Content

CHAPTER 1

INTRODUCTION

Over the past two decades, online social networks (OSNs) such as Twitter, Facebook, YouTube, and Instagram have established themselves as central institutions in the realm of human socialization and interpersonal communication. They have especially disrupted traditional media industries, making online media both more accessible to authors and more reliant on interpersonal connections for visibility [20]. User generated content (UGC) has seen great proliferation under this environment, leading to a flourishing of blogs, videos, and messages all highly interconnected through OSN platforms.

Of particular interest are microblogging platforms such as Twitter, which continue to see widespread adoption and have become active, vibrant communities for online interaction [18]. In the microblogging format, users form connections by "following" other accounts, and post short 140-character messages that are visible to their own followers. For the purposes of this thesis, these messages are also referred to interchangeably as "tweets" or "statuses."[1] Any user may follow and interact with any other user, distinguishing Twitter from more closed social networks like Facebook, which require the "followee" to reciprocate the relationship before the users can interact.

[1]To be precise, we say that all tweets are statuses and all statuses are messages. However, the converse relations do not necessarily hold.

The Twitter microblogging platform has gained particular prominence in the area of real-time news and citizen journalism. Partially due to its public format, UGC on Twitter has the potential to gain viral popularity and achieve far-reaching impact [20]. This also makes Twitter a convenient target for academic study, as the majority of its content is public and freely accessible. However, with messages limited to 140 characters or less, it also presents novel challenges for content-based approaches that rely on larger document lengths.

As microblogging gains traction, the value of becoming an influential participant has become increasingly apparent. Many participants use Twitter as a means to advance their business goals or public image, and treat it as more of a marketing vector than an avenue of self expression [26]. An influential Twitter account can be a highly valued asset for both businesses and individuals [26]. To this end, many users attempt to leverage microblog features, such as hashtags and follow reciprocation, in order to increase their own influence in the system. As microblogs guide more eyes to online content, the study of predicting popularity and influence has become increasingly valuable for the tasks of trend forecasting, studying social dynamics, and predicting real-world events [15]. These efforts commonly fall into the related fields of popularity prediction and influence analysis, both of which we draw from in this work.

1.1 Popularity Prediction and Influence Analysis

Influence analysis and popularity prediction are two distinct but related areas of study in social media. Influence analysis measures the ability of one actor or entity to elicit a change in a social system. This could take the straightforward form

of retweets or favorites, or it could be the more abstract measure of an inferred force within a system. Across the surveyed papers, many different definitions of influence were proposed, using a wide variety of problem formulation. While early influence measures relied largely on relationship graphs between users [19], more recent work has begun to incorporate social media-specific features [9, 17, 37]. These later influence measures add value by offering greater and more specific predictive value than their predecessors. These works on influence analysis often conceptualize influence on a latent feature of users or communities, remaining relatively stable between individual messages [9, 17, 19].

Cha et al. made the famous observation that indegree, or number of followers, is not necessarily a good measure of user influence [11]. They measured influence by a user's propensity to spawn retweets or mentions, and found that influence was gained through a deliberate effort by users, and involved limiting tweets to a single topic. This final point is particularly interesting from our perspective. It provides initial evidence that carefully crafting tweets towards topical content is indeed an effective method of building user influence.

Other approaches rely on more sophisticated structural measures than indegree. TwitterRank is one such paper that proposes a method of quantifying user influence using a modified PageRank algorithm [35]. While these user-based influence studies are somewhat tangential to our study, the methods they use to measure influence can inform our own influence metrics.

Unfortunately, predicting influential tweets is a difficult and unreliable endeavor [7]. Even the measurement of what makes something or someone "influential" is a difficult definition to pin down [12]. In contrast, popularity prediction is a more classic problem formulation, attempting to predict a particular popularity metric over time.

In Twitter, this most often takes the form of retweet events, although favorites and replies could also be considered [15, 16, 33]. These problem formulations are more straightforward, allowing researchers to more easily measure their results. In fact, papers on influence measurement have used popularity prediction as a benchmark to quantify the performance of novel methods [17].

Much of the underlying value of a message's popularity comes from its wide visibility in a network. In fact, for all practical purposes, the visibility of a message and its popularity are equivalent. The primary mechanism by which messages gain visibility on Twitter is by being retweeted, where a user reposts another user's content to their own followers in an attributed manner. In this way, the message reaches followers who may not have seen it originally. Similar additional exposure is given to messages that are favorited or replied to, but with somewhat different social implications. Current popularity prediction methods can be roughly segmented into two approaches. The first attempts to predict popularity as a traditional machine learning task, using various relevant features to make predictions about how much a message will be retweeted [8, 14–16]. Although many relevant factors and methods have been explored, this is still a large, open area of research. Due to the complexity and variability of OSN communities, it remains difficult to predict popularity from traditional tweet features. The second group of prediction methods instead investigate the time dynamics of retweets. By recognizing patterns of retweets over time early in a message's lifecycle, they estimate the total popularity that the message can be expected to achieve [15, 32]. While this has led to effective results, and definitely informs us as to the process by which tweets gain popularity over time, it provides very little information as to the causative factors of a message's popularity.

Within the former predictive methods, many features can be considered. Factors

such as hashtags, URLs, and number of friends and followees have all been shown to be viable predictive features for whether or not a message will be retweeted [33]. However, in the works surveyed, few methods used the content of tweets themselves to predict popularity signals such as retweets. This begs the question of to what degree the message text itself can be used to predict tweet popularity. Particularly, does the topical content of a tweet influence its popularity? To address this question, we can apply modern topic modeling techniques to the twitter dataset in order to correlate topical features with retweet probability. Of course, topic modeling techniques have been applied to related problems such as predicting the adoption rate of hashtags [21] and community-level diffusion extraction [17]. However, applications of topic models in message popularity prediction are surprisingly sparse.

1.2 Topic Models

Topic models provide a quantitative layer for reasoning around natural language documents. Although the qualities and specifics vary from model to model, all rest on the assumption that natural language content pertains to one or more *topics* and that there exists a strong relationship between the content of a document and the topics to which it pertains. Since authors seldom tag their content with topical semantics, topic modelling must both derive meaningful topic representations and accurately infer the topic assignments of content.

One early and ubiquitous topic model, Latent Dirichlet Allocation (LDA), was published in 2003 [10]. LDA is a probabilistic graphical model for the latent topics of a collection of documents. In LDA, topics are represented as multinomial priors distributed on vocabulary, indicating the likelihood of a word occurring given that

particular topic. The key innovation of LDA is to consider that each document could pertain to a mixture of topics, and assign topics not by document but by word. In the inference process, documents are similarly assigned multinomials across topics representing the likelihoods that a given word in the document pertains to a particular topic. While LDA has been demonstrated to work well on a diverse range of documents, it does not do well with the short, colloquial form of Twitter documents [38]. However, Mehrotra et al. [24] propose a method that alleviates the shortcomings of LDA on Twitter without altering the mechanics of LDA. In this study, the authors aggregate tweets into larger documents by pooling them by hashtag. This pooling method leads to an increased coherency in topic models.

Many subsequent topic models have expanded the LDA model by either altering the characteristics of topics or by introducing additional variables that affect topic assignment [17, 29, 31, 34]. Labeled LDA introduces the concept that documents can be assigned labels that correspond to topics [31]. Ramage et al. [29] later apply this system to Twitter by characterizing the topical tendencies of different users. In addition to hashtags, they treat social signals and emoticons as topic labels. This is an interesting direction, but the use of emoticons and social signals is somewhat divergent from this study. Although there are many variations, we focused primarily on topic models that, similar to LDA, represent documents as topic multinomials. This provided a useful representational mapping from word space to a more stable, normalized space, and allowed us to reason quantitatively in this space. It also captured the intuitive insight that although documents may be very different in vocabulary, they can still pertain to similar topics.

LDA is notable partly because it is an unsupervised algorithm. It does not rely on training labels to derive topic vectors or assignments. It is therefore similar to

classical clustering algorithms or unsupervised multilabel classification. In this case, both document classifications and class characteristics are learned properties of the system. Topic weights towards particular words are the class characteristics and one of the primary focuses of LDA analysis. However, they are represented as multinomial vectors with one dimension per word in the corpus vocabulary. Topic vectors therefore instrinsically exhibit a high dimensionality, which must be mitigated for any in-depth analysis.

1.3 Research Aims

In the field of content-based popularity prediction on microblogs, we found little research pertaining to how topic models might be applied to the prediction task. However, there appeared to be a salient link between hashtags, popularity, and topic models [24, 33]. We therefore aimed to address the following questions in our work:

1. How can topic models be applied to popularity prediction?
2. Can hashtag-based popularity prediction techniques be extended to untagged messages using topic modeling techniques?
3. Are there any advantages to using topic models over other popularity prediction techniques?

In the remainder of this work, we discuss our efforts to address these questions. In Chapter 3, we discussed our methods in collecting a large corpus of randomly sampled Twitter messages, and the techniques we used to clean and process this dataset for analysis. In Chapter 4, we explore how hashtags can be used as both a popularity prediction feature and a label for topic vectors. We also measure the

accuracy of a model that uses both of these properties to predict the popularity of untagged messages. Finally, in Chapter 5 we measure the performance of supervised topic models in the popularity prediction task.

CHAPTER 2

THESIS STATEMENT

In this study, we hypothesized that the topical content of a tweet had a correlative relationship with its popularity. Furthermore, we hypothesized that this topical content could be sufficiently captured by LDA-based topic models, and that this correlation could be used to predict the popularity of new content. Our experiments found that while these topic models were successful in prediction tasks, they could be outperformed by other, simpler methods.

CHAPTER 3

DATASET

Dataset collection and storage comprised a large portion of our time spent on this study. Although Twitter messages are publically available, we will show how the collection process itself placed limitations on our sampling methods. We will also show how this affected the nature of our resulting dataset, and what subsequent efforts we made to address perceived limitations.

3.1 Twitter Statuses

The primary dataset collected for this study was five months of Twitter activity extracted from Twitter's public streaming API. Although additional data was gathered by these means, this timespan represented the largest and most contiguous collection run performed by this study. The dataset was large enough to present scalability problems for some data processing stages, so Hadoop was used for data storage and distributed processing, which drastically reduced iteration time in these stages. Using these methods, we were able to collect a nearly-contiguous five-month sample from January through May, 2015, consisting of 2.5T of raw JSON logged from the sample-stream endpoint.

3.1.1 Twitter API

Twitter offers two publicly accessible developer APIs: a REST interface and a Streaming API. Early in this thesis, we investigated the viability of each for our purposes. The REST API offers a comprehensive suite of http endpoints for application developers to interact with Twitter. However, to prevent abuse, all of these endpoints are strictly rate limited at rates that make data collection tasks prohibitively slow. Many rate limits are on a per-user basis, allowing authenticated applications to operate as proxies for users. However, in the case of data collection where there is only one end user, these additional allowed queries are not applicable. Table 3.1 shows a small selection of relevant endpoints and their rate limits [4]. In practice, these rate limits ruled out graph-based analyses, which would have required querying the REST API to obtain friend/follower information on a subset of Twitter users. At 1 paged query per and with users who have thousands of followers, reconstructing a social graph would have been prohibitive.

REST Endpoint	User Auth Requests / Min	App Auth Request / Min
friends/list	1	2
friends/ids	1	1
followers/list	1	2
statuses/lookup	12	6
search/tweets	12	30
users/show	12	12

Table 3.1: Example Twitter REST Endpoint Rate Limits

The Twitter Steaming APIs, while still technically REST endpoints, eschew the classic request/reply model for one where http replies consist of long-running streams of data. There are four categories of streaming APIs: Public APIs, User Streams, Site Streams, and The Firehose [5]. Table 3.2 shows all endpoints exposed in the

Twitter Streaming APIs. Of these, User Streams and Site Streams provide services oriented towards web applications that provide Twitter streams from the perspective of authenticated users. Although we considered establishing collections from the perspective of volunteer users, we ultimately dismissed this approach as impractical from a bureaucratic standpoint. Of the remaining options, the Twitter Firehose returns every status produced on Twitter and requires special access permissions. At over 500 million tweets per day [6], Firehose users need special infrastructure just to receive these data much less store them. Although we initially explored grants and relationships that would allow access to this API, we eventually decided that the engineering challenges surpassed the potential benefit to our project. This left the Public API, which provides the *statuses/sample* and *statuses/filter* endpoints. The Status Sample endpoint provides a straightforward "small random sample of all public statuses" [2]. The Status Filter returns public statuses that match one or more filters. These filters could be set over user IDs, keywords, or locations, under the constraints specified in Table 3.3. Due to these limitations, we chose to build a dataset from the Status Sample API. As detailed in Section 3.1.2, the Status Sample API provided a large volume of data to work with.

Streaming Endpoint	Endpoint Type
statuses/sample	Public API
statuses/filter	Public API
user	User Streams
site	Site Streams
statuses/firehose	Firehose

Table 3.2: Twitter Streaming Endpoints

Filter	Limit
User IDs	5000 Users
Keywords	400 Words
Location	25 0.5–360 Degrees

Table 3.3: Filter Limits on *statuses/filter*

3.1.2 Collection

From the outset, our ambition was to collect a terabyte-scale dataset of Twitter messages in order to counteract the high dimensionality and sparsity of text data. To this end, our collection and storage platform was the *bdserver* Hadoop cluster consisting of one name node and 5 data nodes with a post-redundancy capacity of 26.9T. See Appendix A.1 for details on this machine and its configuration. This allowed us ample storage for not only the primary dataset but any files produced by subsequent data processing jobs. The MapReduce and YARN frameworks provided application-level tools for authoring distributed data processing jobs.

In order to facilitate collection, we authored the Java application twitter-fh to run for long periods of time on a server with access to a Hadoop file system. In addition to its collection capabilities, twitter-fh includes a publisher-subscriber architecture allowing it to be configured for logging, as well as multiple concurrent storage media and formats. Because Twitter aggressively cuts off multi-account stream subscriptions from the same machine, it was imperative that twitter-fh be powerful enough to handle multiple non-trivial stream subscriptions, particularly to the filter stream. This extensible architecture allowed it to be easily modified when we needed to add additional behavior such as binned data storage and more complex stream subscriptions.

After its completion, twitter-fh daemon was then run on the bdserver name node beginning in July 2014. Technical issues and bugfixing prevented it from running

continuously, so it intermittently produced collections for the rest of 2014. This was less of a problem than one might think, as we still had ample data to work with from its initial collection runs. Rather than prioritizing 100% uptime, we chose to focus on data exploration of the initial output with the intention of executing a longer collection run when additional data was necessary. We observed a rough average of 20G per day in data volume, which we deemed sufficiently large for late-stage collection. In January 2015, full-time collection was resumed in earnest after fixing some of the technical issues with the collection program. Figure 3.1 illustrates the collection timeline for our data. Due to its relative continuity, most experiments were run considering only the 2015 dataset from January 6th to May 28th. These tweets were then separated and archived on the HDFS file system.

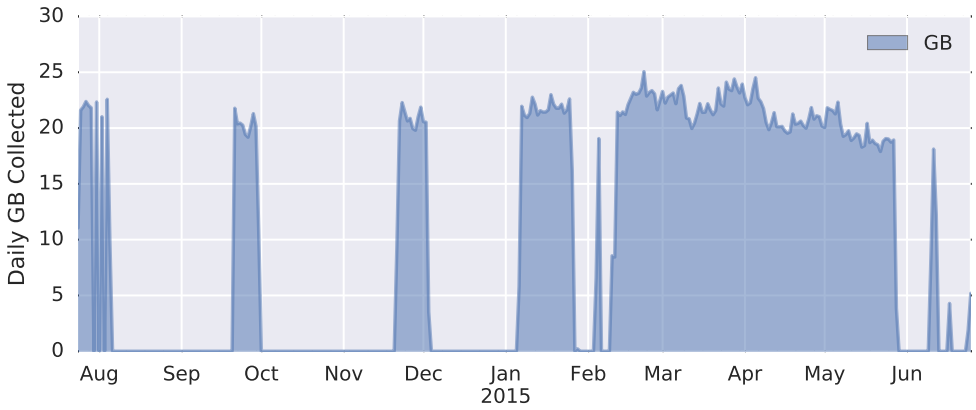

Figure 3.1: Daily Collection Volumes for 2014 through 2015

3.2 Processing Techniques

When we began the data exploration phase of this thesis, we struggled to find a one-size-fits-all solution for data management. Particularly in the case of LDA-based PGMs, many of the algorithms necessary for our experiments only had single-threaded

reference implementations. In general, these implementations were not designed for even gigabyte-scale datasets, and did not generalize well on terascale data. To reimplement many of them in a massively parallel context would be a separate thesis itself. On the other hand, YARN and MapReduce tools allowed us to easily collect simple statistics such as mean and standard deviation across our full dataset. In order to both leverage our large dataset and explore more sophisticated PGMs, we opted for a multi-stage data pipeline where YARN applications performed the dual tasks of statistics collection and dataset cleaning. These big data jobs produced much smaller, heavily filtered datasets suitable for the more sophisticated single-machine analysis algorithms. An added benefit of this approach was that with an overabundance of data, we had the luxury of discarding noisy or unhelpful sections of data that threatened to impede the performance of our topic models.

3.2.1 Big Data

For the initial data processing stages, we made heavy use of the Cascading data processing framework [1]. This open source framework provided a layer of abstraction on top of YARN and MapReduce, allowing us to develop data operations in terms of discrete "nodes," which were then composed into a minimal number of MapReduce jobs. This paradigm allowed us to rapidly compose multiple MapReduce stages into one comprehensive data flow, and made large-scale data processing far more manageable and efficient. Via either aggregation, filtering or sampling, these data flows would produce cleaned datasets small enough for use in single-machine analysis.

Although dataset cleaning and processing was an ongoing, iterative task throughout the thesis, our final dataset discussed below is the most comprehensive in terms of text cleaning, and produced many of the final results described later in this thesis

Unlike earlier dual-role processing tasks, this task was created for the sole purpose of creating a cleaned corpus suitable for topic modeling. For the sake of brevity, we refer to this task as the *TClean* task, and its resulting dataset as the *TClean* dataset. Another, earlier dataset was the *InfluenceFlow* dataset, which performed aggregations and transformations to produce many derivative dataset features, while performing less aggressive filtering on the resulting corpus. As we will discuss in Chapter 4, the resulting size of this dataset eventually proved to be too unwieldly.

InfluenceFlow

The InfluenceFlow dataset was an initially simple tweet processing pipeline made with the intent of reformatting tweets for use with an L-LDA implementation, which grew over time to accomodate the various needs of different investigations. Although it performed many tasks, InfluenceFlow was organized around the principle of maximal data retention. Therefore, beyond very basic tweet cleaning stages the corpus was minimally filtered. Table 3.4 shows a number of statistics from different stages of the InfluenceFlow data processing task.

Metric	Count
Raw Tweets	88,509,696
Cleaned Tweets	25,402,948
Hashtagged Tweets	4,691,545
Training Tweets	4,691,319
Testing Tweets	226
Unique Hashtags	1,012,176
Unique Users	9,913,571
User Interactions	19,143,165

Table 3.4: InfluenceFlow Dataset Statistics

TClean

The TClean dataset was produced at a late stage in this thesis, as the result of many earlier lessons learned. Earlier experiments had shown the significant scalability challenges of running new topic models on our dataset. Although large-scale implementations existed for common models such as LDA, these were highly specific and difficult to generalize to other candidate models such as LLDA, sLDA, and COLD. However, simply truncating the data to the first few million tweets would run the risk of producing a dataset where few similarities could be found across sparse language features. Instead, TClean was a dataset created to be a smaller, more manageable tweet corpus while still leveraging the increased sample size of the larger 2.5T collection.

A key insight was that LDA and its derivatives applied a statistical model to bag-of-word documents, and therefore, words which only occurred once contributed negligibly to the final model. Instead, stable models would extract co-occurence patterns between words and documents in the form of topics. A word could only be identified as a significant topic indicator if it ocurred relatively frequently in the corpus.

Therefore, after thorough text cleaning steps, TClean went through a dual filtering stage where infrequent words were filtered out from all documents, and then documents falling below a given word count were further removed. By filtering "long-tail words" in this way, we obtained a subset of Twitter documents biased towards the modal vocabulary of the corpus. Not only was the dataset reduced to a manageable size, but the document vocabulary was guaranteed to frequently co-occur within the corpus. In this way, we were able to restrict the scope of our problem to only consider

tweets containing "sufficiently typical" vocabulary. The full cleaning pipeline can be found in Table 3.5. Note that this process relies on two threshold values, t_1 and t_2, which filter out words and documents based on word frequency and document length.

Cleaning Step	Description
Language Filter	Remove any non-english tweets using metadata
Deletion Notices	Remove any tweets which are later deleted
Lowercase	Cast all tweet text to lowercase characters
Character Filtering	Remove punctuation and numerals
Tokenization	Split tweets into words
Stopwords	Remove topically neutral stopwords
Stemming	Stem words using the Porter Stemmer[28]
Word Frequency Filter	Remove words which occur less than $t_1 = 200$ times
Document Length Filter	Remove documents with a length less than $t_2 = 5$

Table 3.5: TClean Cleaning Steps

To select appropriate word count threshold, we ran a tangential data flow which calculated word counts as well as the maximum threshold which would allow each document $d \in D$ to remain in the corpus. We denote these value as s_w^V and s_d^D respectively. We then calculated the number of occurrences for each value, denoted $f_n^\star$ in Equations 3.1 and 3.2. This created an output dataset small enough to analyze on a traditional single-machine setup. Here a simple transformation gave us $r_n^\star$ in Equations 3.3 and 3.4, which expresses the number of words and documents, respectively, that would remain in the corpus for $t_1 = n$. With these values, we could accurately evaluate the impact of threshold t_1 on both the word and document dimensionality in the resulting dataset.

$$f_n^V = \left|\{\, w \in V \mid s_w^V = n \,\}\right| \tag{3.1}$$

$$f_n^D = |\{\, d \in D \mid s_d = n \,\}| \tag{3.2}$$

$$r_n^V = \sum_{i \geq n} f_i^V \tag{3.3}$$

$$r_n^D = \sum_{i \geq n} f_i^D \tag{3.4}$$

Upon investigation, we found that t_1 had a much more pronounced effect on remaining word count r^V than it did on remaining document count r^D. Figure 3.2 depicts r^V across varying scales of t_1, while Figure 3.3 depicts the similar relationship of r^D and t_1. Both cases demonstrate a clear power law distribution, each function decreasing exponentially. We can also see that for threshold values as high as 20,000, the number of remaining documents is still over 70,000,000, while the number of remaining words has dropped to 5,000. This is consistent with our understanding of the corpus, since we can imagine that no matter how much we restrict the vocabulary, there will always be more tweets.

We therefore chose a value for t_1 that constrained vocabulary size rather than document count. Our reasoning was that the excessively high t_1 required to reduce the number of documents to a small size would allow for only a trivial vocabulary. Instead, we chose to set $t_1 = 200$, as shown in Table 3.5. Figure 3.2 shows that this corresponds to a vocabulary size of roughly 100,000. We expected this size to be computationally tractable, while still allowing for sufficient word variation. Using this threshold value, we generated the final TClean dataset, which we refer to in the rest of this work. Table 3.6 shows the breakdown of tweets in various stages, indicating that TClean is approximately 12.5% of the original dataset's size, with a 43% reduction in the threshold filtering stage.

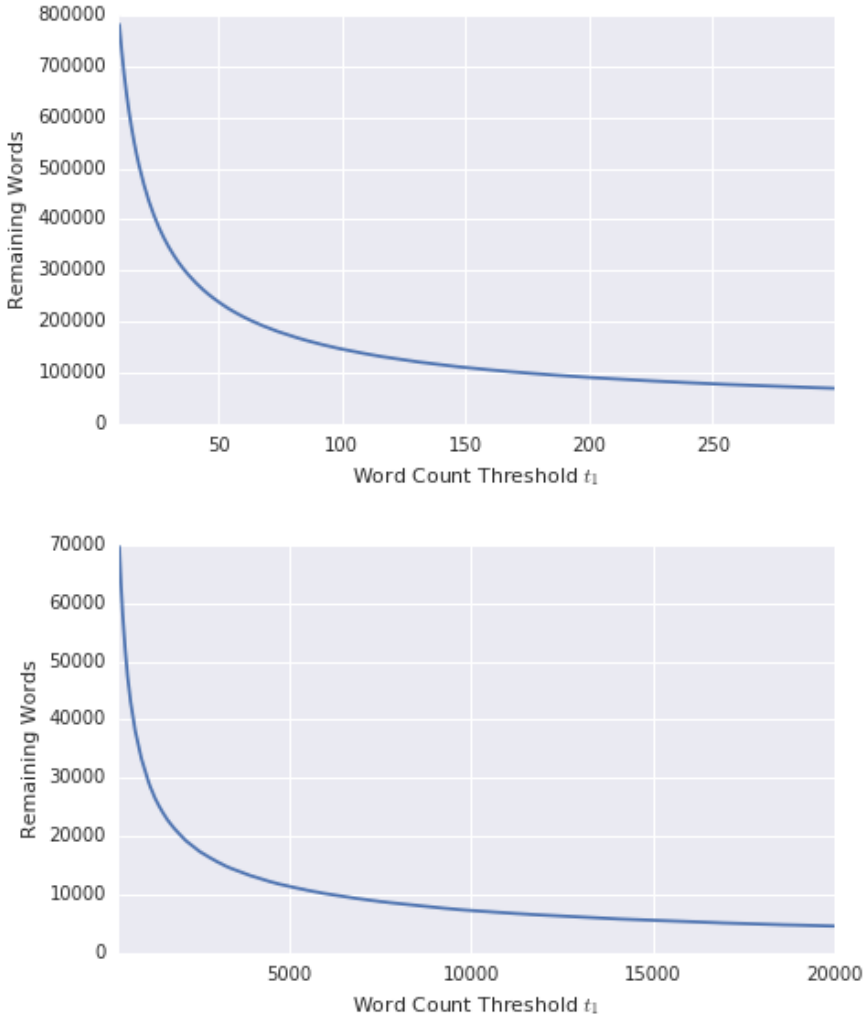

Figure 3.2: Effect of t_1 Threshold on Remaining Word Count r_n^V on the domains $t_1 \in [15, 300]$ and $t_1 \in [3 \times 10^2, 2 \times 10^4]$

Cleaning Stage	Count
Input Tweets	703,558,103
Language Filter	172,432,743
Deletions	149,876,675
Deletion Filter	155,154,459
Threshold Fiter	88,443,396

Table 3.6: TClean Cleaning Statistics

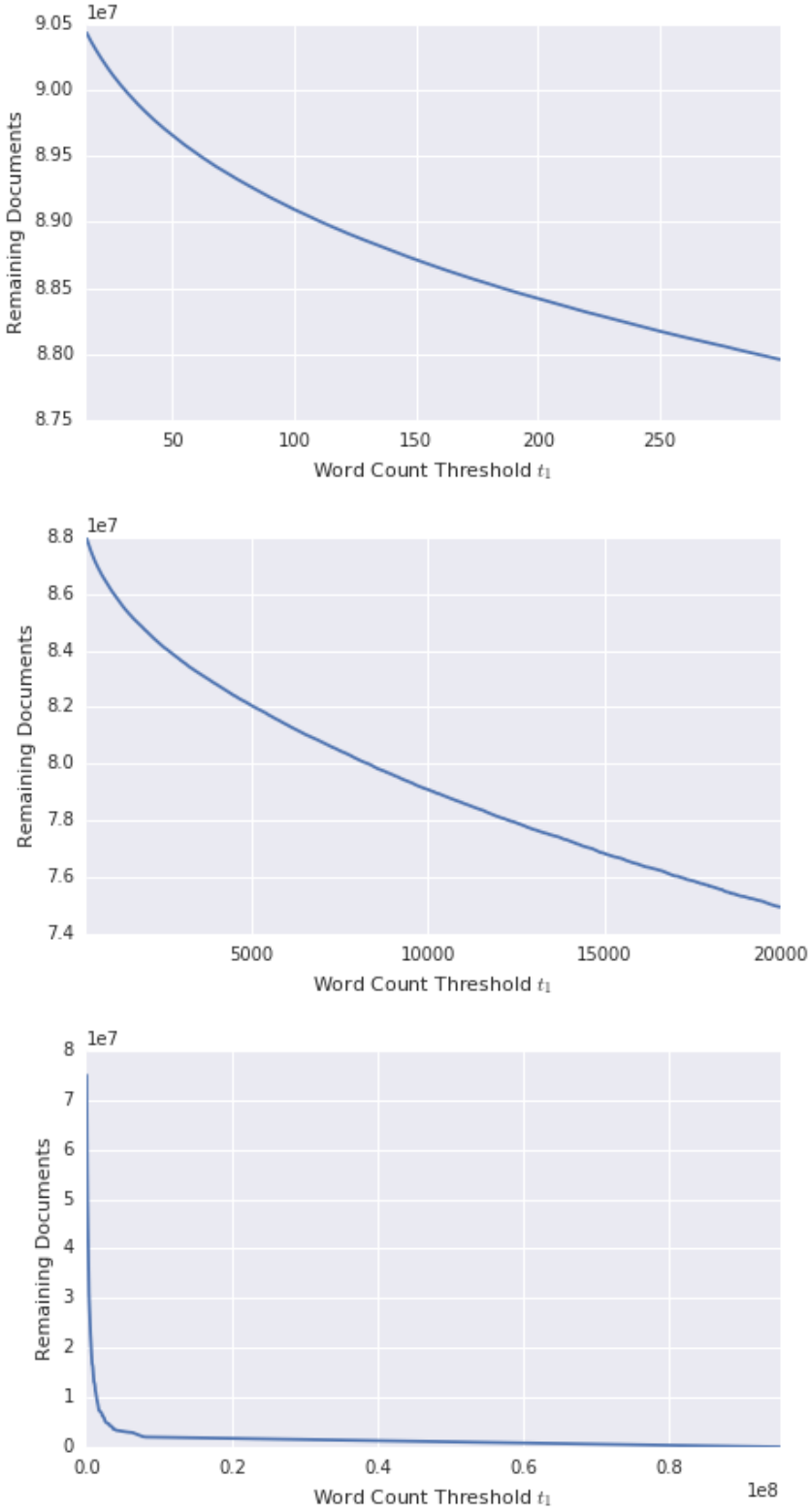

Figure 3.3: Effect of t_1 Threshold on Remaining Document Count r_n^D on the domains $t_1 \in [15, 300]$, $t_1 \in [3 \times 10^2, 2 \times 10^4]$, and $t_1 \geq 2 \times 10^4$

3.2.2 Local Analysis

After obtaining a filtered local dataset such as TClean, we needed to run a wide variety of data exploration and analysis tasks, using tools written in a wide variety of languages. To facilitate and organize these tasks, we used a minimal, flexible convention of separating data transformation tasks into separate subfolders, each with a Makefile and symlinks that would use environment variables to perform its data analysis task on the correct symlinked dataset. Although somewhat arcane in comparison to other more sophisticated build tools, this allowed us the flexibility to use multiple, different tools without constraining ourselves to a particular technology stack. From here, the final stages of these data flows were typically ipython notebooks, which illustrated the results for analysis. This was often an iterative process, with our ipython reports prompting questions which needed additional steps to our data flow to investigate.

CHAPTER 4

HASHTAGS AND LLDA FOR POPULARITY PREDICTION

Content on Twitter often encorporates special "hashtag" tokens, which are words prepended with the octothorpe "#" symbol (e.g., "#PorteOuverte" or "#YesAllWomen"). The web platform then converts these hashtags into hyperlinks to pages that display collected feeds of all tweets containing a particular hashtag. In this way, users can follow or participate in larger conversations by labeling their content with relevant hashtags. Hashtags can be interpreted as intuitive topic labels for content, curated by the entire microblogging community.

It has also been observed that hashtags play a dual role in social media. In addition to annotating content with similar topics, hashtags serve as a way for users to identify with a community. Yang et al. [36] demonstrated that this dual role can be used to effectively predict future adoption of hashtags by users based on their membership in a community. This work showed that hashtags symbolize not just topics but audiences around that topic [36]. Additionally, Suh et al. [33] have found evidence that hashtags "have a strong relationship with retweetability" [33]. With our hypothesis that retweetability could be predicted by a tweet's topical content, we investigated the use of hashtags as features for retweet prediction.

We sought to confirm the relationship between hashtags and retweets by formu-

lating a predictive model that considered *only* hashtags for popularity prediction. If an effective model could be established, then it could be extended with topic models in order to draw the link between text content and hashtags. However, the first step was to confirm that hashtags were indeed effective features for popularity prediction, and to formulate a predictive model based on these isolated features.

4.1 Using Hashtags for Popularity Prediction

To model the relationship between hashtags and retweets, we formulated our inputs as a classic binary prediction problem. Given a tweet's hashtags as input features (X), we endeavored to predict whether or not it would be a retweet (y) as a binary categorical variable. For the purposes of our initial study, we restricted the prediction problem to only those tweets that contained at least one hashtag in the InfluenceFlow dataset. As mentioned in Section 3.2.1, this constituted approximately 4.7 million tweets, or roughly 19% of the cleaned dataset.

Before we can discuss the input features for this model, we must first formalize how this problem is defined. First, for N documents, let $D = [1, N]$ be a range of identifying numbers that can be bijected onto the set of documents. This allows for a convenient handle with which to refer to the documents, as well as an implied ordering for any matrix that contains information pertaining to the document. Similarly, for the vocabulary of M unique hashtags within the corpus, let $H = [1, M]$ be a range of identifying numbers for these hashtags. In order to indicate whether a tweet uses a hashtag, let $I^{\#}_{dh}$ be the indicator for whether document $d \in D$ uses hashtag $h \in H$ as defined in Equation 4.1. Similarly, let I^{R}_{d} be the indicator for whether $d \in D$ is a retweet, as defined in Equation 4.2. Finally, we will often need to refer to the subset

of all tweets that use a particular hashtag, or all hashtags that a particular tweet uses. We refer to these with H^D and $D^\#$ as defined in Equations 4.3 and 4.4, respectively.

$$I_{dh}^{\#} = \begin{cases} 1 & \text{if } d \in D \text{ uses } h \in H \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

$$I_{d}^{R} = \begin{cases} 1 & \text{if } d \in D \text{ is a retweet} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

$$H_d^D = \left\{ h \in H \;\middle|\; I_{dh}^{\#} = 1 \right\} \tag{4.3}$$

$$D_h^{\#} = \left\{ d \in D \;\middle|\; I_{dh}^{\#} = 1 \right\} \tag{4.4}$$

With this formal structure in place, we sought to understand the probability that a message would be retweeted given its use of a hashtag. We denote this as $p_h^{\#}$ from Equation 4.5. Note that since hashtags are the only feature considered in this model, this probability is assumed to be invariant between documents as per Equation 4.6. Therefore, we do not index $p^{\#}$ by any document $d \in D$. From here we estimate $p^{\#}$ from the observed data as $\hat{p}^{\#}$ using the mean from observed data as shown in Equation 4.9. This was a quantity that could easily be derived from our full dataset using MapReduce. We also define the counting functions C^R and $C^{\not R}$ in Equations 4.7 and 4.8 for use in defining $\hat{p}_h^{\#}$ as well as later measures.

$$p_h^{\#} = \Pr\left(I_d^R = 1 | I_{dh}^{\#} = 1\right) \tag{4.5}$$

$$(\forall i, j \in D)\left(\Pr\left(I_i^R = 1 | I_{ih}^{\#} = 1\right) = \Pr\left(I_j^R = 1 | I_{jh}^{\#} = 1\right)\right) \tag{4.6}$$

$$C_h^R = \sum_{d \in D_h^{\#}} I_d^R \tag{4.7}$$

$$C_h^{\not R} = \left|D_h^{\#}\right| - C_h^R \tag{4.8}$$

$$\hat{p}_h^{\#} = \frac{C_h^R}{\left|D_h^{\#}\right|} \tag{4.9}$$

First, we note that $\hat{p}^{\#}$ already represents a simple prediction model for whether a tweet will be a retweet. However, due to its formulation as a conditional probability, it is only suitable for tweets that use exactly one hashtag. For tweets that use more than one hashtag, we therefore take the mean of $\hat{p}_h^{\#}$ across all $h \in H_d^D$ to derive the metric $\hat{p}^A$ described in Equation 4.10. When we apply this to our test set from the InfluenceFlow corpus, which contains 206 tweets with one or more hashtags, we can measure its performance as a predictor. To measure classification performance, we use two common metrics: log loss ($\mathcal{L}$) as defined in Equation 4.11, and area under the receiver operating characteristic curves ($\mathcal{A}$ or AUC) as defined in Equation 4.12. While the former aims to demonstrate a reliable error for probability estimates of binary classifications, the latter showcases the performance of a model across all possible classification thresholds. The ideal score for a classifier would be $\mathcal{L} = 0$ and $\mathcal{A} = 1$. The performance of $\hat{p}^A$ under these metrics is shown in Table 4.1, along with its ROC curve in Figure 4.1.

$$\hat{p}_d^A = \frac{1}{|H_d^D|} \sum_{h \in H_d^D} \hat{p}_h^\# \tag{4.10}$$

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{N} \sum_{n=1}^{N} [y_n \log \hat{y}_n + (1 - y_n) \log (1 - \hat{y}_n)] \tag{4.11}$$

$$\mathcal{A}(y, \hat{y}) = \int_0^1 \mathrm{TPR}_T(y, \hat{y}) \mathrm{FPR}'_T(y, \hat{y}) dT \tag{4.12}$$

$$\mathrm{TPR}_T(y, \hat{y}) = \text{True positive rate of } \hat{y} \text{ using classification threshold } T \tag{4.13}$$

$$\mathrm{FPR}_T(y, \hat{y}) = \text{False positive rate of } \hat{y} \text{ using classification threshold } T \tag{4.14}$$

Metric	Score
Log Loss ($\mathcal{L}$)	1.620970
ROC AUC ($\mathcal{A}$)	0.806827

Table 4.1: $\hat{p}^A$ Classification Performance

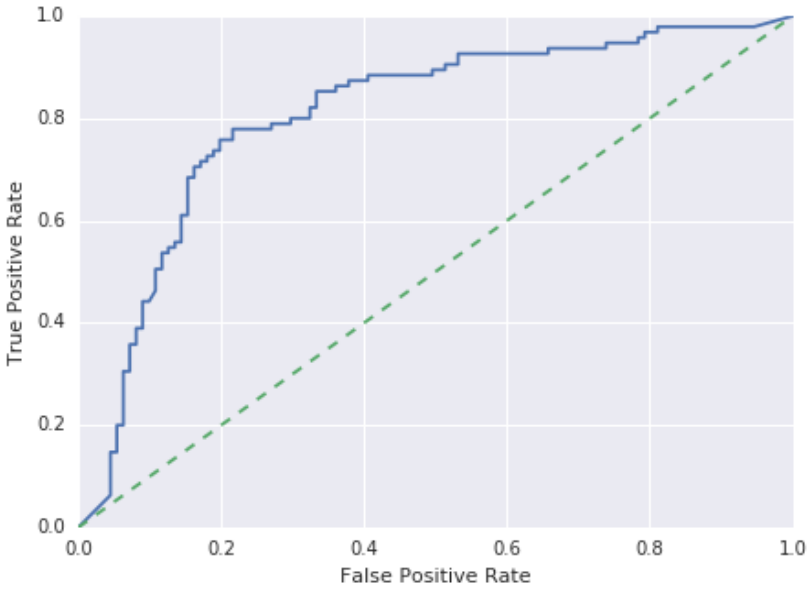

Figure 4.1: $\hat{p}^A$ Classification ROC Curve

Having established $\hat{p}^A$, we trained a logistic classification model using the *scikit-learn* python library [13, 25]. Due to the large size of our dataset, it was impractical to train this model on each individual tweet. Instead, we relaxed our training algorithm to treat each unique hashtag as two weighted training points: one for positive retweet evidence and one for negative retweet evidence. Equations 4.15 through 4.20 express the training features of such a model. Note that for a given hashtag $h \in H$, the two observations X_{2h} and X_{2h-1} take the same value of $\hat{p}_h^{\#}$, while their y values are 1 and 0, respectively. Weighting is then used to represent that hashtag's proportional usage in the dataset. This model formulation is equivalent to creating one observation of $\left(\hat{p}_h^{\#}, I_d^R\right)$ for every hashtag h in every document d, and then training our logistic classifier as normal. However, by using weights precomputed by a MapReduce task, we can drastically reduce the dimensionality of our dataset without any loss in accuracy.

$$X_{2h} = \hat{p}_h^{\#} \tag{4.15}$$

$$X_{2h-1} = \hat{p}_h^{\#} \tag{4.16}$$

$$y_{2h} = 1 \tag{4.17}$$

$$y_{2h-1} = 0 \tag{4.18}$$

$$w_{2h} = C_h^R \tag{4.19}$$

$$w_{2h-1} = c_h^{\not R} \tag{4.20}$$

We can see a visualization of the trained model in Figure 4.2. Here the blue dots depict training points with sizes corresponding to their weights, while the classifier outputs for different values of $\hat{p}^A$ are depicted by the green line. Table 4.2 shows the

performance metrics for this classifier on the InfluenceFlow test set, while Figure 4.3 shows its receiver operating characteristic. Here it is evident that while the log loss has improved significantly, the AUC remains unchanged. This is consistent with our understanding of the logistic regression, since our classifier can be represented as a bijection between two monotonically increasing functions. As ROC analysis measures the performance of any possible classification threshold, every threshold on the original curve would be mapped to exactly one threshold point on the classifier's predictions.

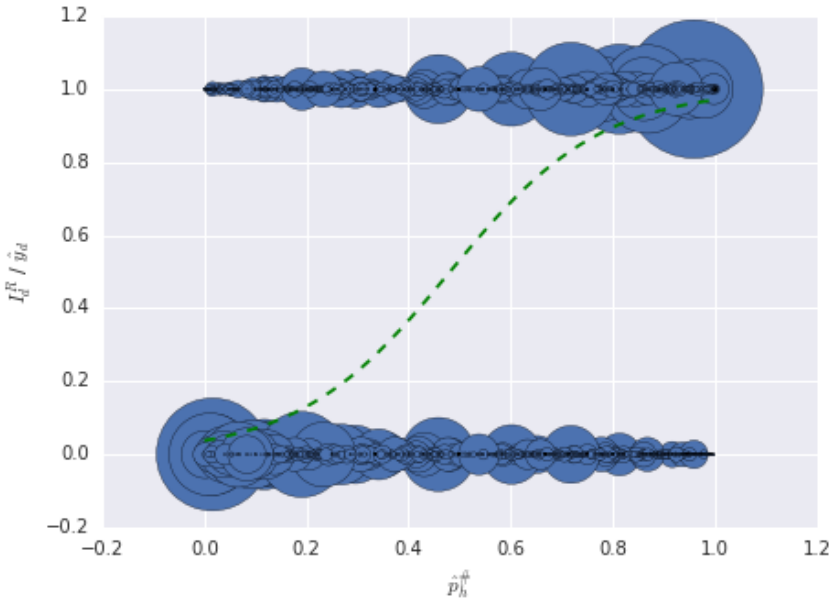

Figure 4.2: $\hat{p}^A$ Logistic Classifier Visualization

Metric	Score
Log Loss ($\mathcal{L}$)	0.576261
ROC AUC ($\mathcal{A}$)	0.806827

Table 4.2: $\hat{p}^A$ Logistic Classifier Performance

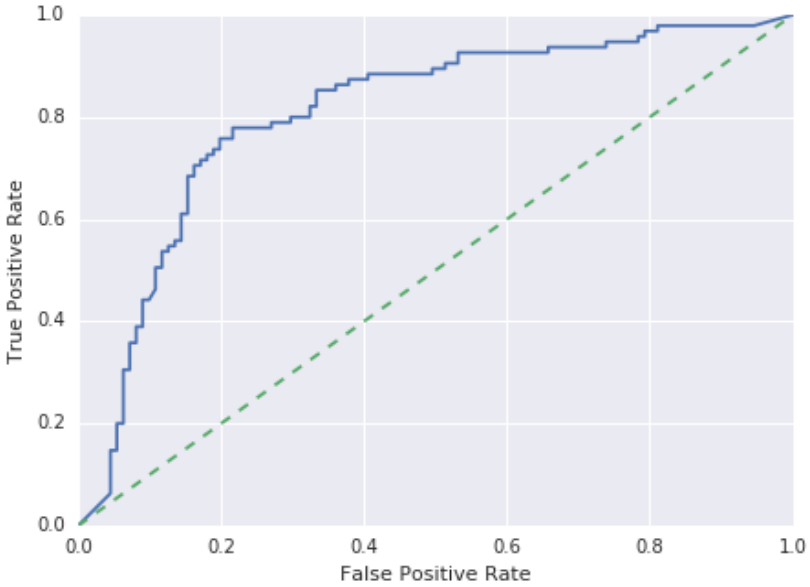

Figure 4.3: $\hat{p}^A$ Logistic Classifier Receiver Operating Characteristic

We also investigated a second probability estimator $\hat{p}^B$ as defined in Equation 4.21. $\hat{p}^B$ is similar to $\hat{p}^A$, but weights the mean of $\hat{p}_h^{\#}$ by the number of occurrences of hashtag h. Our aim was to create an estimator that was weighted by the evidence available for each input feature. However, as shown in Tables 4.3 and 4.4, it was categorically outperformed by $\hat{p}^A$. We therefore gave it very little further attention. Rather than requiring readers to reread this chapter for comparisons, we have included a summary of scores in Table 4.5. Here we see that the logistic classifier on $\hat{p}^A$ is the strongest predictor among surveyed methods.

$$\hat{p}_d^B = \frac{\sum_{h \in H_d^D} \left(\hat{p}_h^{\#} \left| D_h^{\#} \right| \right)}{\sum_{h \in H_d^D} \left| D_h^{\#} \right|} \tag{4.21}$$

Metric	Score
Log Loss ($\mathcal{L}$)	1.650664
ROC AUC ($\mathcal{A}$)	0.783594

Table 4.3: $\hat{p}^B$ Clasification Performance

Metric	Score
Log Loss ($\mathcal{L}$)	0.624463
ROC AUC ($\mathcal{A}$)	0.783594

Table 4.4: $\hat{p}^B$ Logistic Classifier Performance

X	Method	$\mathcal{L}$	$\mathcal{A}$
Baseline	$\hat{y} = E[I_d^R]$	0.693637	0.500000
$\hat{p}^A$	$\hat{y} = X$	1.620970	0.806828
$\hat{p}^A$	Logistic Classifier	0.585102	0.806828
$\hat{p}^B$	$\hat{y} = X$	1.650664	0.783594
$\hat{p}^B$	Logistic Classifier	0.624463	0.783594

Table 4.5: Summary of Hashtag Predictor Performance Metrics

4.2 Identifying the Topic Spaces of Hashtags

After measuring the efficacy of hashtags as popularity predictors, we conducted a series of experiments investigating the possibility of mapping hashtags to topic vectors and vise versa. We hypothesized that by mapping content to topic-similar hashtags, we could make popularity predictions on untagged content that outperformed the baseline. We investigated LLDA as well as TF-IDF as mechanisms for performing this mapping, but ultimately found them both to be intractable for our purposes.

4.2.1 Labeled Latent Dirichlet Allocation

We have previously noted the role of hashtags as explicit topic labels for content. Labeled Latent Dirichlet Allocation (LLDA) is a topic model that accomodates explicit topic labels, and has previously been successfully applied to recommendation tasks on microblogs [30, 31]. Using hashtags as topic labels, we investigated the use of LLDA in mapping hashtags to topic vectors. The advantage of this approach was that under the LLDA model, every hashtag maps to a single topic vector. Therefore, by estimating the topic distribution of an unlabeled document, its hashtag mapping would be made explicit by the model.

LLDA extends the LDA model by adding document labels that have a one-to-one correspondence with topics, and if a document has one or more labels, then the document may only draw words from the corresponding topics. Unlabeled topics are unrestricted and behave the same as in the LDA model. This generative model for LLDA is expressed in Equations 4.22 through 4.28. Figure 4.4 shows the same generative model in plate notation.

$$\beta_k \sim \text{Dir}(\eta) \qquad \forall k \in K \tag{4.22}$$

$$\Lambda_{d,k} \sim \text{Bernoulli}(\Phi) \qquad \forall d \in D, \forall k \in K \tag{4.23}$$

$$L_{ij}^{(d)} = \begin{cases} 1 & \text{if } \Lambda_{di} = j \\ 0 & \text{otherwise} \end{cases} \qquad \forall d \in D \tag{4.24}$$

$$\alpha^{(d)} = L^{(d)} \times \alpha \qquad \forall d \in D \tag{4.25}$$

$$\theta_d \sim \text{Dir}\left(\alpha^{(d)}\right) \qquad \forall d \in D \tag{4.26}$$

$$z_{d,i} \sim \text{Mult}\left(\theta_d\right) \qquad \forall n \in N_d, \forall d \in D \tag{4.27}$$

$$w_{d,n} \sim \text{Mult}\left(\beta_{z_i}\right) \qquad \forall n \in N_d, \forall d \in D \tag{4.28}$$

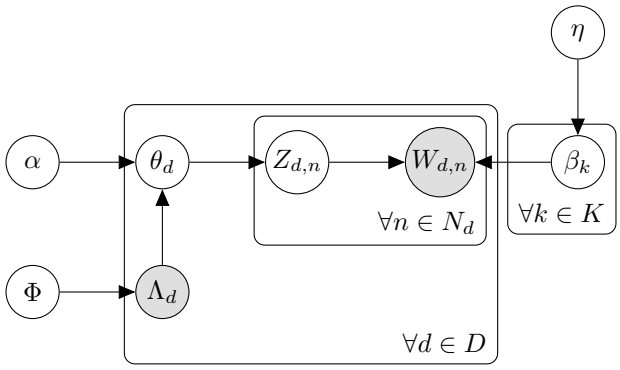

Figure 4.4: A Plate Notation Representation of LLDA [31]

For our purposes, we used the open source *JGibbLabeledLDA* implementation to apply LLDA using hashtags as labels [3]. The JGibbLabeledLDA project is a single-core implementation of LLDA that uses collapsed Gibbs sampling for model estimation [27]. We also made use of *JGibbLDA*: a similar implementation of LDA, which also used collapsed Gibbs sampling. Our aim was to estimate an LLDA model from the full InfluenceFlow training dataset, possibly by running LLDA over the course of months. It is important to note that under our training regime, there would

exist one topic per unique hashtag, which would have an impact on the dimensionality of our estimated θ and β vectors. Due to the large scale of the training dataset, we expected this to be a large and long-running task.

Initial runs of the program encountered memory usage exceptions, indicating that the program was unable to scale to our full-sized dataset. All experiments were run on the *infolab* server (As discussed in Appendix A.2) in order to accomodate these large memory requirements. We therefore scaled back our approach, and ran JGibbLabeledLDA and the related JGibbLDA to explore its scalability properties. To control runtime, we created two truncated datasets of 5×10^3 and 1×10^5 tweets, respectively. Table 4.6 shows the runtimes and memory usage of running JGibbLDA and LGibbLabeledLDA on these datasets. We found that for datasets larger than 1×10^5, runtimes became unmanageable and excessive memory usage often caused exceptions during execution. We attributed this result to the increased scale introduced by mapping hashtags to topics, and therefore having one topic for every unique hashtag. As shown in Table 4.6, the number of topics increases by two orders of magnitude. We can expect the peak memory usage to similarly expand to at least 400 GB, which is pushing the abilities of even our high-capacity infolab server. Furthermore, 1×10^5 represents only a small fraction of our dataset, indicating that our LDA and LLDA implementations would be poorly equipped to train on the full InfluenceFlow corpus.

We concluded that since the upper training limit of 1×10^5 represented 0.3% of cleaned tweets in our InfluenceFlow corpus, and 2.1% of all hashtagged tweets in the same, it was unlikely that LLDA would be an effective tool for mapping content to hashtags in the larger corpus. Instead, we shifted focus to simpler methods that could rely on values calculated in MapReduce.

Records	Algorithm	# Topics	Time	Peak Memory Usage
5×10^3	LDA	100	2m 16s	1.69 GB
5×10^3	LLDA	1093	15m 17s	28.19 GB
1×10^5	LDA	100	45m 36s	4.00 GB
1×10^5	LLDA	11432	48h 49m 24s	Unrecorded

Table 4.6: Run Times for LDA and LLDA tests

4.2.2 Token Correlation

In order to formulate a scalable mapping from content to hashtags, we used a variation of term frequency inverse document frequency (TFIDF) [22]. Equation 4.32 shows a TFIDF measure where rather than using tweets as documents, we instead use hashtags. Equations 4.29 and 4.30 support this measure with relevant definitions of counting values C^D and $C^\#$. We use $C^\#$ to express hashtags as documents consisting of all words they co-occur with. Equation 4.33 provides a similarity measure between hashtags and documents. This was used in Equation 4.34 to derive the estimator $\hat{p}^C$.

$$C^D_{d,w} = \#\text{ of times word } w \text{ occurs in document } d \tag{4.29}$$

$$C^{\#}_{h,w} = \sum_{d \in D^{\#}_h} C^D_{d,w} \tag{4.30}$$

$$V^D_d = \{\, w \in V \mid C^D_{d,w} \geq 1 \,\} \tag{4.31}$$

$$\mathit{tfidf}(w,h) = \frac{C^{\#}_{h,w}}{\sum_{v \in V} C^{\#}_{h,v}} \times \frac{|H|}{\left|\left\{\, h \in H \;\middle|\; C^{\#}_{h,w} \geq 1 \,\right\}\right|} \tag{4.32}$$

$$\mathit{sim}(d,h) = \frac{1}{|t|} \sum_{w \in V^D_d} \mathit{tfidf}(w,h) \tag{4.33}$$

$$\hat{p}^C_d = \frac{\sum_{h \in H} \left(\hat{P}^{\#}_h \times \mathit{sim}(d,h) \right)}{\sum_{h \in H} \mathit{sim}(d,h)} \tag{4.34}$$

We then applied the measurement techniques from Section 4.2.1 to $\hat{p}^C$. First we measured the performance of $\hat{p}^C$ as a predictor, and then that of the logistic classifier from Section 4.1 when $\hat{p}^C$ was applied as input. Tables 4.7 and 4.8 show the results of these measurements, while Figure 4.5 shows the corresponding ROC curve. Note that as in Section 4.1, the ROC curves are identical. Therefore, only one is included here. Furthermore, a prediction performance summary is again shown in Table 4.9.

Metric	Score
Log Loss ($\mathcal{L}$)	0.679541
ROC AUC ($\mathcal{A}$)	0.648648

Table 4.7: $\hat{p}^C$ Clasification Performance

Metric	Score
Log Loss ($\mathcal{L}$)	0.765525
ROC AUC ($\mathcal{A}$)	0.648648

Table 4.8: $\hat{p}^C$ Logistic Classifier Performance

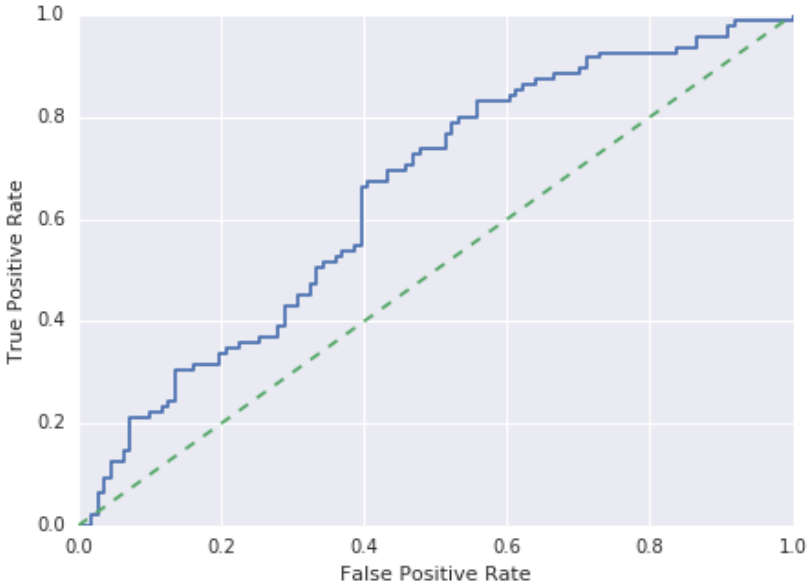

Figure 4.5: $\hat{p}^C$ Logistic Classifier ROC Curve

X	Method	$\mathcal{L}$	$\mathcal{A}$
Baseline	$\hat{y} = E\left[I_d^R\right]$	0.693637	0.500000
$\hat{p}^A$	$\hat{y} = X$	1.620970	0.806828
$\hat{p}^A$	Logistic Classifier	0.585102	0.806828
$\hat{p}^B$	$\hat{y} = X$	1.650664	0.783594
$\hat{p}^B$	Logistic Classifier	0.624463	0.783594
$\hat{p}^C$	$\hat{y} = X$	0.679541	0.648648
$\hat{p}^C$	Logistic Classifier	0.765525	0.648648

Table 4.9: Summary of Hashtag Predictor Performance Metrics

In these tables, first observe that for $\hat{y} = X$, $\hat{p}^C$ drastically outperforms $\hat{p}^A$ and $\hat{p}^B$. Although this was an interesting result, we could not derive any meaningful conclusions from it. As far as we can tell, it is little more than an interesting artifact of unfitted data. Much more legible were the results showing that $\mathcal{L}$ and $\mathcal{A}$ of the logistic classifier are significantly worse for $\hat{p}^C$ than that of the $\hat{p}^A$ or $\hat{p}^B$. However, $\hat{p}^C$ still outperforms the baseline of $\hat{y} = E\left[I_d^R\right]$. From this, we concluded that hashtags can be used to predict the popularity of content, but that when available, using hashtags directly would result in better performance.

Together, the results from Table 4.9 showed that the predictive capacity of hashtags could be successfully leveraged when using content to predict popularity. However, the resulting predictor suffered a loss in performance, most likely due to the noise introduced in the mapping. Knowing this, we next aimed to compare the performance of this predictor to other content-based methods. However, before we could continue, we would need to revisit our dataset in order to construct a manageable corpus for algorithms that had no available MapReduce implementation.

CHAPTER 5

DIRECT POPULARITY PREDICTION WITH SLDA

After concluding our investigation into the use of LLDA on popular hashtags, we shifted our focus to the potential of using another variant of LDA called Supervised Latent Dirichlet Allocation (sLDA) [23], which integrates characteristics of both topic modeling and a supervised learning task. We found this configuration to be an ideal candidate for a popularity prediction algorithm on topical features due to its integration of topic models with the supervised learning problem.

Supervised Latent Dirichlet Allocation is a statistical model introduced in collaboration with the primary author on LDA, motivated by previous attempts by researchers to apply LDA to supervised learning tasks [23]. These applications address problems that can be broadly categorized as cases where text documents have an associated response variable that must be predicted. Many of these prior attempts used LDA-based topic models as input features for their regression methods, similar to our own attempts with LLDA. In contrast, the sLDA model integrates document response variables with the topic model itself. This allows for the estimation of topic vectors that are fitted not only to their content, but to the response variable itself.

Integration of the response variable is achieved by formulating a model where the response is a random variable conditioned on a document's estimated topics. This formulation is general enough to accomodate a variety of response types by

applying a suitable distribution in the model. For example, if the response variable is categorical, a multinomial distribution could be used, while a real-valued response could be modeled with a gaussian distribution. However, both the original paper and our reference implementation described sLDA where the response is a normally distributed variable, so we will focus on that case here.

In such a case where the response variable $y \in \mathbb{R}$, sLDA takes the model parameters described in Table 5.1. With these parameters established, each document and response is generated as follows:

1. For each document $d \in D$
   (a) Draw topic distribution $\theta_d \sim \text{Dir}(\alpha)$.
   (b) For each word $n \in N$
      i. Draw topic assignment $z_n|\theta \sim \text{Mult}(\theta)$.
      ii. Draw word from topic $w_n|z_n, \beta_{1:K} \sim \text{Mult}(\beta_{z_n})$.
   (c) Draw response variable $y|z_{1:N}, \eta, \sigma^2 \sim N\left(\eta^\top \bar{z}, \sigma^2\right)$.

$$\bar{z} = \sum_{n=1}^{N} z_n \tag{5.1}$$

In this process, $\bar{z}$ is the weighted average between drawn topics, as defined in Equation 5.1. This generative process is illustrated in plate notation by Figure 5.1, providing a convenient graphical representation.

Knowing this model, we hypothesized that sLDA could be used to predict retweets based on message content. To this end, we formulated a study where we used an sLDA predictor on our TClean dataset, using reweets as our response variable, and measured

Parameter	Description
K	Number of topics.
V	Number of unique words.
α	The document-topic dirichlet parameter
$\beta_{1:K}$	The topic vectors, each a β_k being a V-dimensional multinomial distribution. In our reference implementation, these are themselves estimated from a dirichlet prior in the same fashion as an LDA model.
η	A vector of response means where η_k is the mean response for topic k.
σ	A vector of response deviations where σ_k is the standard deviation for topic k.

Table 5.1: Model Parameters for sLDA

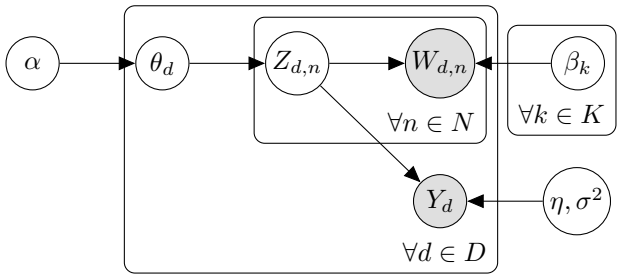

Figure 5.1: A Plate Notation Representation of sLDA [23]

the efficacy of this approach. Our experiments with sLDA centered around running it on our TClean dataset, with a document's retweet status as its signal variable.

Rather than reimplementing sLDA for our experiments, we sought to use a reference implementation to save time and effort. Although we considered the original implementation released by Mcauliffe and Blei [23], we ultimately used the R implementation found in the "lda" R package on CRAN for its speed and usability. Whereas the original implementation performs its estimation via variational inference, the R sLDA implementation instead uses collapsed Gibbs sampling.

After our experiences with LLDA, we deemed the challenge of running sLDA on the full-scale dataset to be out of scope of our project, and instead invested our time on creating the new *TClean* dataset described in Section 3.2.1. It is important to note that in order to achieve data reduction, TClean aggressively filters content from the original dataset down to content with a "representative" vocabulary, meaning that any results we arrive at are for a particular kind of tweet. Specifically, the results of this study describe the performance of sLDA on tweets that satisfy the vocabulary frequency constraints described in Section 3.2.1, which corresponds to a tweet using a sufficient "median vocabulary" of tokens that occur more frequently than most within the corpus.

For our experiments, we formulated the prediction problem as a machine learning task. The features X and response variables y were document word counts and a retweet indicator, respectively, as described in Equations 5.2 and 5.3.

$$X_{dw} = \#\text{ of occurences of } w \text{ in } d \tag{5.2}$$

$$y_d = \begin{cases} 1 & \text{if } d \text{ is a retweet} \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

In these experiments, we aimed to answer the following questions:

1. What sLDA parameters led to the best retweet predictions?
2. How did these predictions compare to our hashtag-based model in Chapter 4?
3. Were these predictions sufficiently accurate to indicate a correlation between tweet topics and retweets?

5.1 Parameter Selection for SLDA

For the purposes of our experiment, our ultimate goal was a configuration of sLDA that produced predictions as accurately as possible. However, our reference implementation took multiple tuning parameters, some of which had large impacts on the ultimate running time of the process. Other parameters had little impact on running time, but would need to be adjusted to maximize performance. We first investigated parameters with high impacts on time performance to see if we could observe trends in speed and performance. From there, we used our results to run longer sLDA runs using the knowledge we had gained from shorter runs.

Altogether the reference sLDA implementation took 7 tuning parameters, as described in Table 5.2. Of these, the variables α, η, β, and σ could all be roughly estimated from model averages, and had no impact on the running time of the model estimation. The remaining three, m, e, and K, all had an impact on the running

time, in addition to the dimensionality of the dataset itself. While e and m modulated the number of gibbs sampling sweeps and the number of expectation maximization iterations, respectively, k controlled the number of latent topics that would be used in the model. These were the variables we chose to focus on in our preliminary runs, so that we could gather performance data that could later inform more expensive computations.

Parameter	Description
$K, \alpha, \eta, \beta, \sigma$	Model parameters. See Table 5.1.
e	The number of Gibbs sampling sweeps to make over the entire corpus for each iteration of EM.
m	The number of EM iterations to make.

Table 5.2: Estimation Parameters for sLDA

5.1.1 Sweep 1E5–1

In order to investigate these relationships, we first ran a series of parameter sweeps on our sLDA implementation. These initial sweeps focused on parameters that impacted the running time of the model, such as e, m, k, which run on a relatively small dataset. In each sweep, we measured both the runtime of our sLDA fitting (t) as well as its predictive performance, hoping to establish metrics on both. For our performance metric, we chose the log-loss metric ($\mathcal{L}$) from Equation 4.11, as it is a natural fit for a binary response variable and a probabilistic prediction. For convenience, we have restated its formula in Equation 5.4 below.

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \tag{5.4}$$

$$\hat{y}_d = \text{Predicted } y_i \text{ given } X_i \tag{5.5}$$

First, we truncated the TClean dataset down to its first 10^5 tweets, denoting this new dataset TClean5. This allowed us to run sLDA estimations in a shorter amount of time, which seemed reasonable as the original authors used datasets with even fewer documents in their original paper [23]. After truncation, TClean5 was split into randomly sampled training and testing sets, at a ratio of 80% training records to 20% testing. Training records were further segmented into 3 equally sized validation folds ($F = 3$), again using random sampling.

The sweep itself was then performed across the range of values described in Table 5.3. For each iteration in the sweep, a sLDA model would be cross-validated over the validation folds using parameters from the range in Table 5.3, as well as the static parameters described in Table 5.4. We measured both the time elapsed while estimating the model, as well as the log-loss prediction performance of the estimated model on the held-out validation fold. With 3 validation folds, this allowed us to take the mean and standard deviation of each parameter set, in order to confirm that our measurements were representative for a given parameter set. Table 5.5 provides a summary of these measurements, as well as the derived values used in later analysis. All runs were performed on the Sweet Chedda machine discussed in Appendix A.3.

Parameter	Start	End	Step Size
e	10	80	10
m	2	10	2
k	10	40	10

Table 5.3: Ranges for sLDA Sweep 1E5–1

Parameter	Value
F	3
α	$1/K$
β	$1/V$
η	Retweet signal mean in training set
σ^2	Retweet signal variance in training set

Table 5.4: Static Parameters for sLDA Sweep 1E5–1

Measure	Description
L^f_{emk}	Log loss for sLDA model on training fold f with parameters e, m, and k. See Equation 5.4.
$\hat{L}_{emk}$	Mean log loss for sLDA model with parameters e, m, and k. See Equation 5.6.
σ^L_{emk}	Standard deviation of log loss for sLDA model on training fold f with parameters e, m, and k. See Equation 5.7.
Err^L_{emk}	Estimated 3σ error of $\hat{L}_{emk}$ as a percentage of its value. See Equation 5.8.
t^f_{emk}	Elapsed seconds for training sLDA model on training fold f with parameters e, m, and k.
$\hat{t}_{emk}$	Mean elapsed seconds for sLDA model with parameters e, m, and k. See Equation 5.6.
σ^t_{emk}	Standard deviation of elapsed seconds for sLDA model on training fold f with parameters e, m, and k. See Equation 5.7.
Err^t_{emk}	Estimated 3σ error of $\hat{t}_{emk}$ as a percentage of its value. See Equation 5.8.

Table 5.5: Measurements from sLDA Sweep 1E5–1

$$\hat{L}_{emk} = \frac{1}{F} \sum_{f \in F} L^f_{emk} \tag{5.6}$$

$$\sigma^v_{emk} = \sqrt{\frac{1}{F} \sum_{f \in F} \left(v^f_{emk} - \hat{v}_{emk} \right)^2} \tag{5.7}$$

$$\mathrm{Err}^v_{emk} = 100 \times \frac{3\sigma^v_{emk}}{\hat{v}_{emk}} \tag{5.8}$$

The measured effects of e, m, and k on $\hat{L}$ and t are summarized in Figure 5.2, which illustrates $\hat{L}_{emk}$ and $\hat{t}_{emk}$ for all permutations of e, m, and k. Figure 5.3 illustrates the corresponding values of Err^{L}_{emk} and Err^{t}_{emk}. From these figures, we drew the following key observations:

1. Almost all Err_{emk} error bars fell below 3% of their values, indicating that there would likely be little variation between estimation runs in relation to measured values. Outliers were primarily for low values of $\hat{t}_{emk}$.

2. Despite this, the magnitude of $3\sigma_{emk}$ is often greater than the differences between $\hat{L}_{emk}$ and $\hat{t}_{emk}$ for different values of input parameters. This would suggest that it is not uncommon to have observations where one set of parameters outperforms another, even when its average performance would be worse. In other words, due to the variations between folds, the advantage of one parameter set over another is only measurable over multiple folds.

3. For $k = 10$, $\hat{L}_{emk}$ indicated little visual correlation with any values of e or m. However, as k increased to higher values, $\hat{L}_{emk}$ displayed a negative correlation with both e and m values, particularly for the higher values of $k = 30$ and $k = 40$.

4. $\hat{L}_{emk}$ also seemed to take lower values for higher k.

Although all of these observations were subjective and qualitative interpretations of the data, together they provided early evidence suggesting that our hypothesis was correct, and sLDA could be used to predict retweets to at least some measurable degree. Furthermore, they indicated that e, m, and k all had a positive predictive impact on the system. However, the cost of increasing any of these variables was the

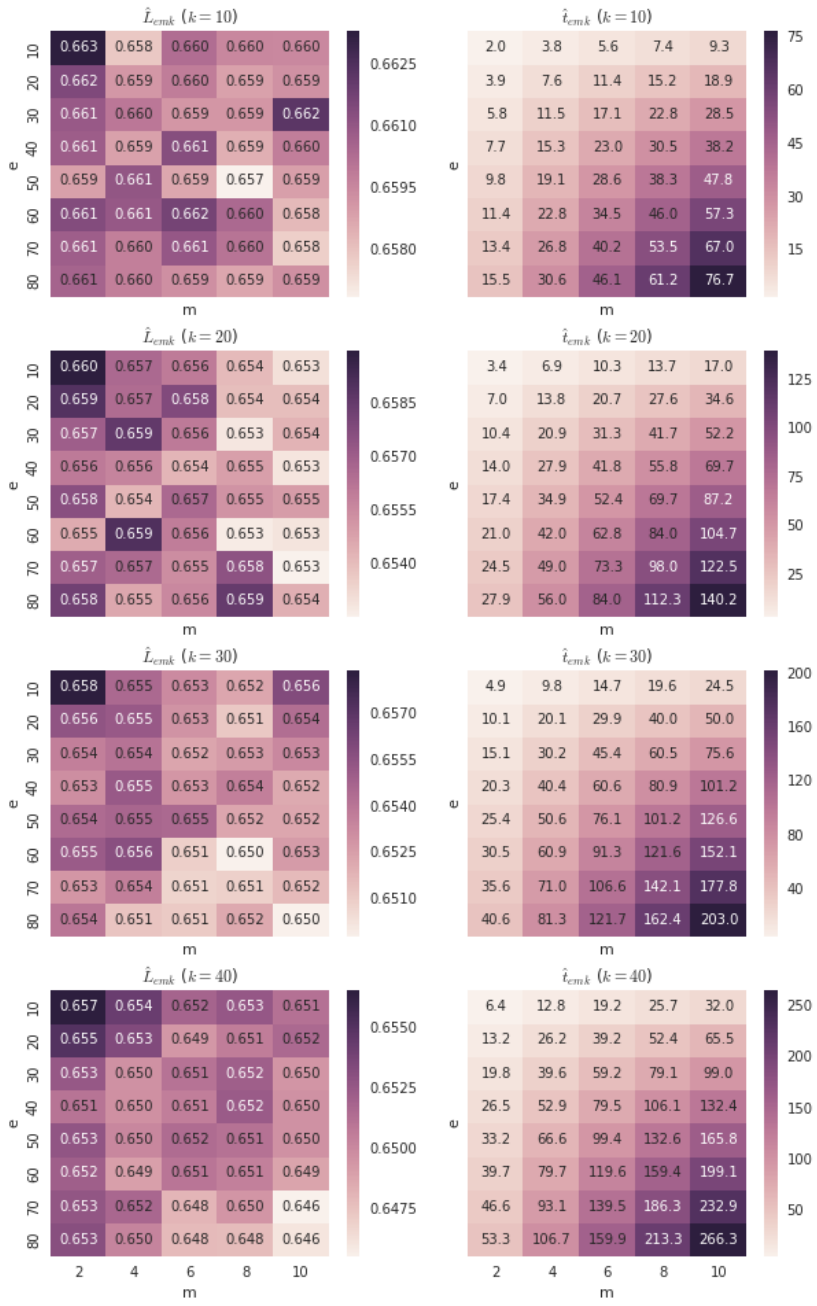

Figure 5.2: Sweep 1E5–1 Mean Log Loss and Mean Elapsed Time

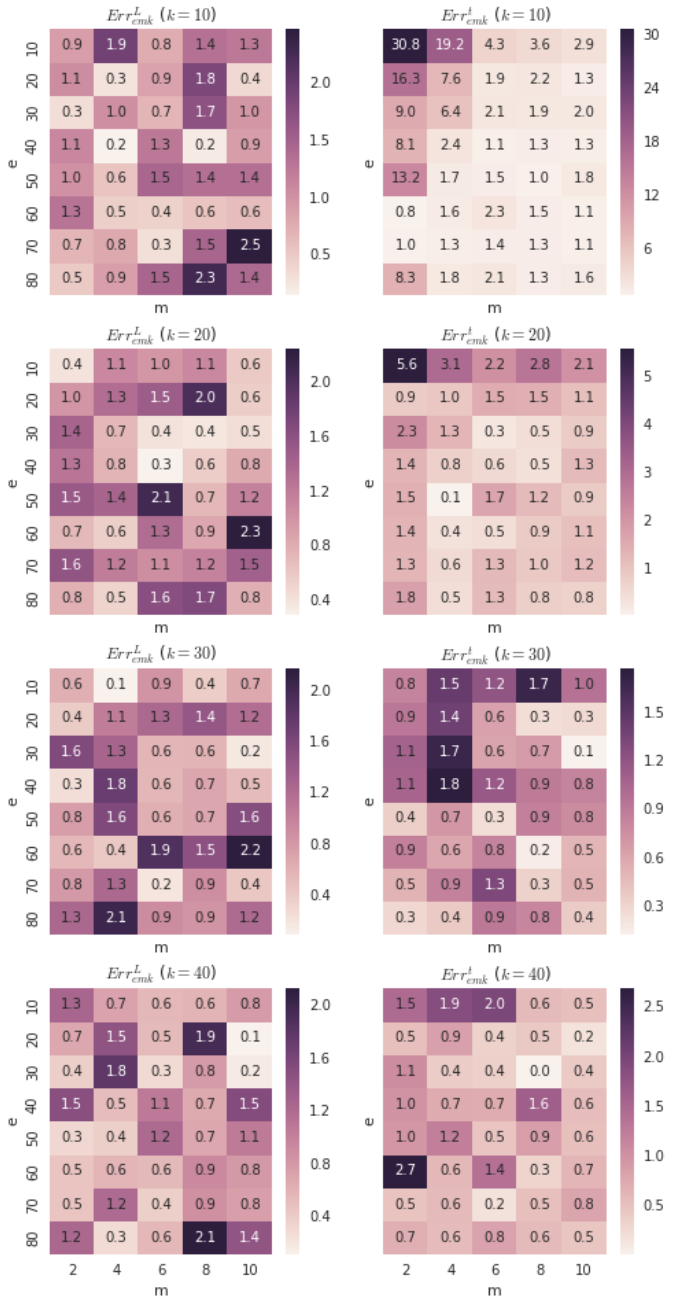

Figure 5.3: Error % of Sweep 1E5–1 Mean Log Loss and Mean Elapsed Time

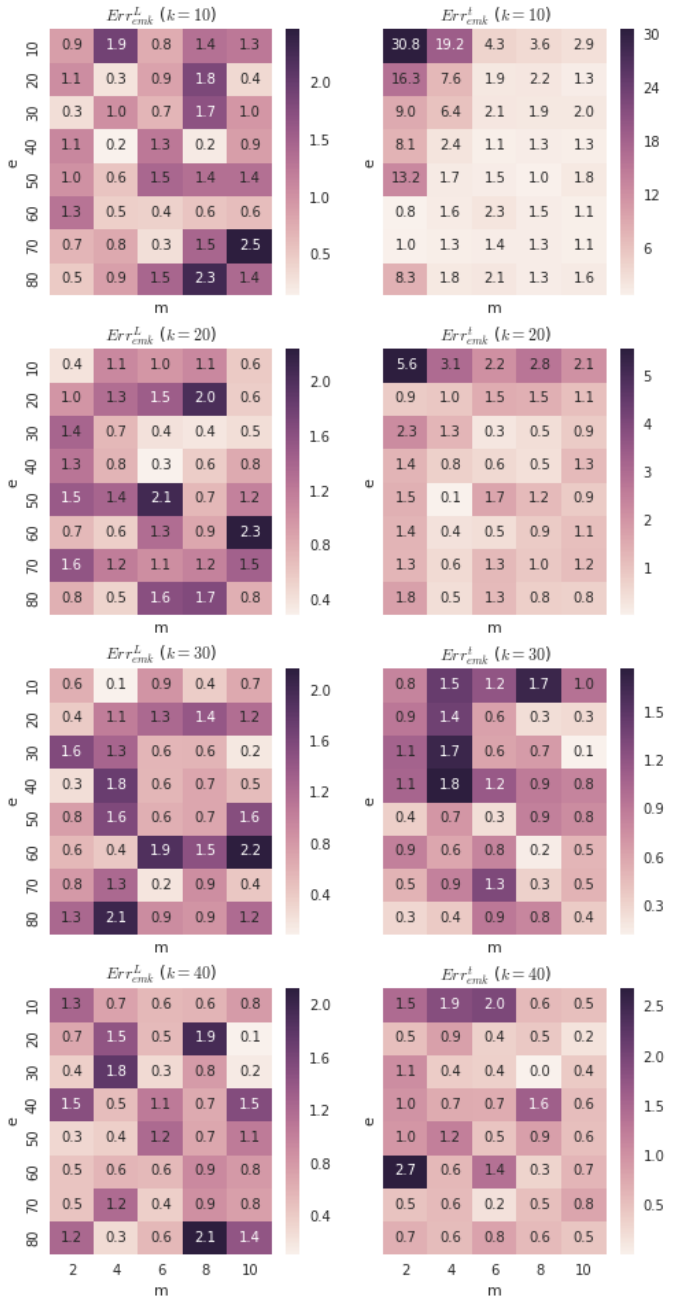

Figure 5.4: $3\sigma_{emk}$ of Sweep 1E5–1 Mean Log Loss and Mean Elapsed Time

increased time required to estimate the corresponding model. Therefore, we aimed to discover the combinations of e, m, and k that would yield optimal prediction performance for a given timeframe.

Before exploring what such an optimal combination would look like, we first took some time to establish the relationship between e, m, k, and $\hat{t}_{emk}$. If we were to vary any one of these constants, holding the others fixed, we would expect the time complexity to increase linearly. Therefore, we hypothesized that $\hat{t}_{emk} \approx c \times e \times m \times k$, for some $c \in \mathbb{R}_{\geq 0}$. However, since this was a minor corrolary to our study, we chose to validate this empirically rather than performing a complexity analysis. To this end, we derived the dimensionality measure Dim_{emk} defined in Equation 5.9 and performed a linear regression between Dim_{emk} and $\hat{t}_{emk}$, in addition to measuring the Pearson correlation coefficient. The results are listed in Table 5.6, as well as displayed visually in Figure 5.5. With a correlation coefficient of $r \approx 0.999155$, we were confident in the linear relationship between Dim_{emk} and $\hat{t}_{emk}$. This would be useful later, when we wanted to run sLDA estimations for a particular duration.

$$\text{Dim}_{emk} = e \times m \times k \tag{5.9}$$

Parameter	Value
Slope	0.008330
Intercept	1.448136
RValue	0.999155
PValue	0.000000
StdErr	0.000016

Table 5.6: Sweep 1E5–1 Dimensionality vs Timing Regression Results

Having firmly established the relationship between our model parameters and

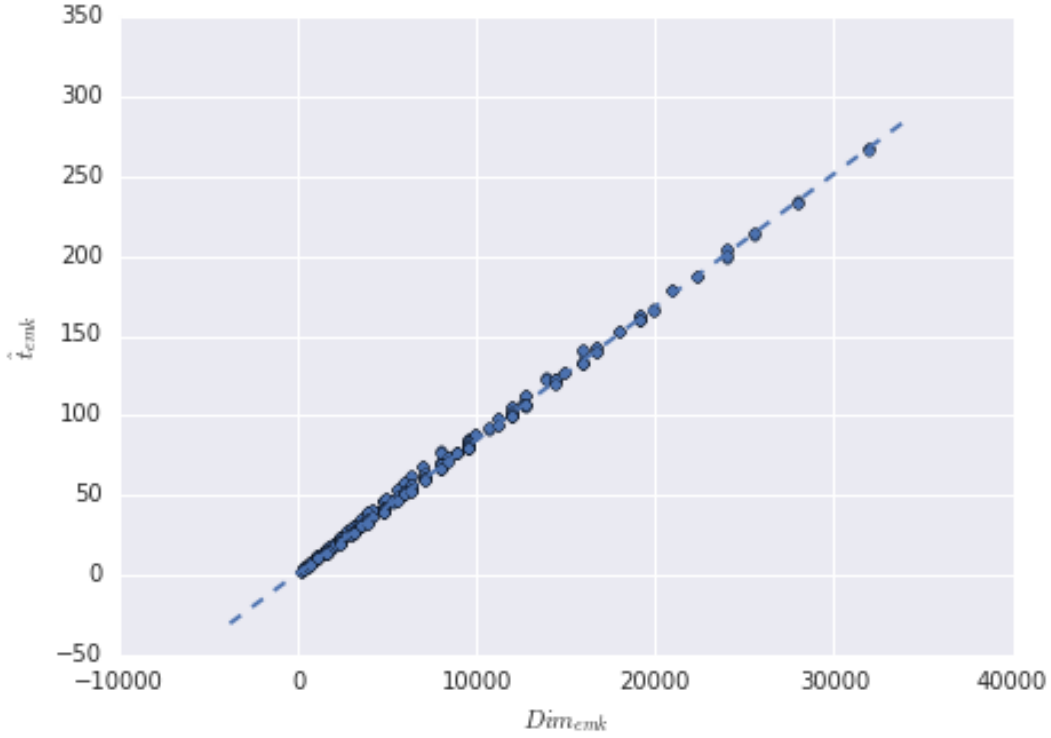

Figure 5.5: Sweep 1E5–1 Dimensionality vs Timing Regression

sLDA estimation time, we then turned to examining the more ambiguous relationship between these variables and $\hat{L}_{emk}$. Figure 5.6 provides an entry point for this relationship, suggesting the trend that as we spend more time estimating our model, the log loss of its predictions will decrease. Seeking to investigate this relationship further, Figure 5.7 displays the same scatter plot with logistic regression lines overlayed for different values of e, m, and k. Although this is far from a conclusive solution, we can observe that while regression lines for e and m are somewhat disordered, those for the k plot show a clear anticorrelation between k and $\hat{L}_{emk}$. Even for measurements of roughly the same duration, a higher k seems to correlate to better performance in the model. We found this relationship to be less qualitatively obvious in our explorations of e and m. Figure 5.8 deconstructs Figure 5.7 further by separating measurements by their k value. Here we can see that trendlines for k decrease in slope as k increases. This would suggest that not only is a higher k better, but it has a higher potential for performance gains as the model runs longer.

It is important to note that the previous observations are all qualitative in nature. Although these trends are highly suggestive, they do not alone indicate any sort of optimal combination of e, m, and k, nor the tradeoffs associated with sacrificing one for another. After numerous attempts, we still struggled to tease out this deeper relationship between $\hat{L}_{emk}$ and e, m, and k. We eventually decided to perform a second sweep, using what we knew about the time complexity of our problem to run another sweep across e, m, and k that held time constant.

5.1.2 Sweep 1E5–2

Sweep 1E5–2 was run as a followup to Sweep 1E5–1, using much of the same methodology. Like Sweep 1E5–1, it was a parameter sweep that ran 3-fold cross-validations

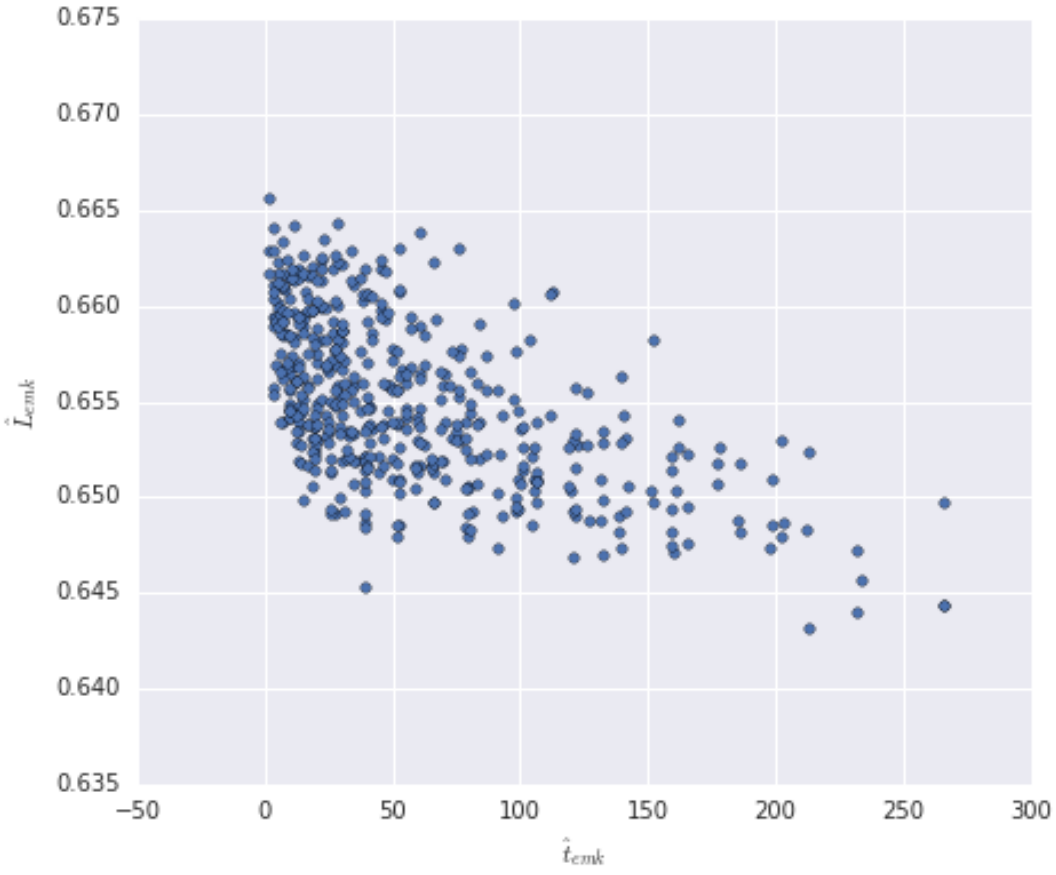

Figure 5.6: Sweep 1E5–1 Log Loss vs Elapsed Time

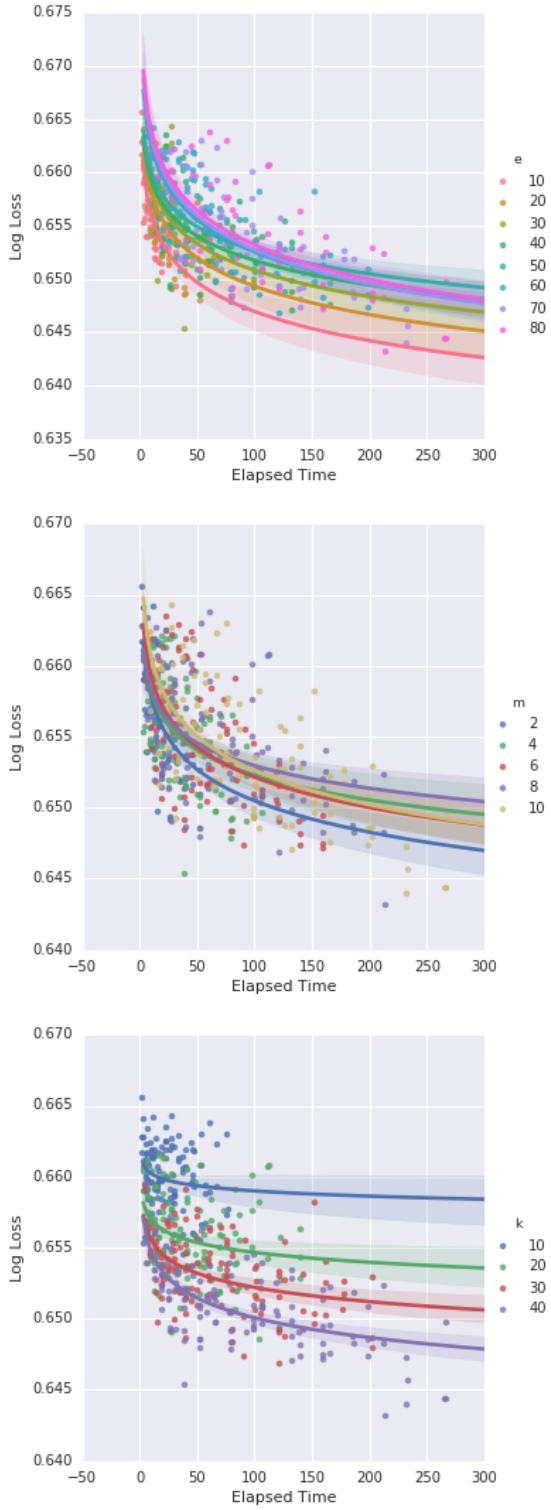

Figure 5.7: Sweep 1E5–1 Log Loss vs Elapsed Time, Stratified by e, m, and k

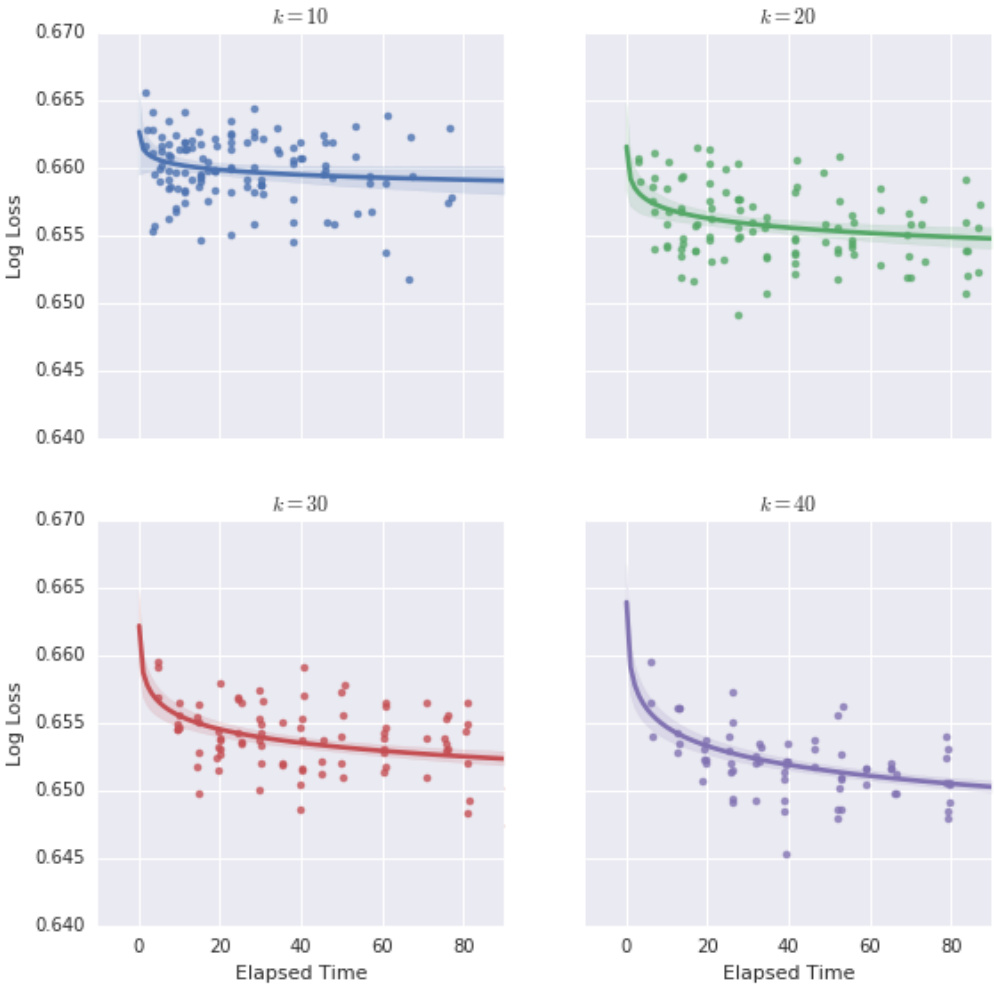

Figure 5.8: Sweep 1E5–1 Log Loss vs Elapsed Time Split By $k \in K$

of sLDA models parameterized on e, m, and k. However, Sweep 1E5–2 used our knowledge of $\hat{t}_{emk}$ from Section 5.1.1 to select values for e, m, and k, which had the same runtime by holding Dim_{emk} constant. Having confirmed the correlation between Dim_{emk} and $\hat{L}_{emk}$, we hoped that by isolating it from the equation we could shed some light on the interplay between e, m, and k in the same time context.

In this sweep, we chose to select Dim_{emk} approximately corresponding to runtimes of $\hat{t}_{emk} \in \{180, 300, 420\}$ (3–7m). This led to Dim_{emk} taking the values in Table 5.7, which also describes the sweep ranges for m and k. The e variable was fixed by the other time sensitive parameters, as described in Equation 5.10 (in order to satisfy the relationship defined in Equation 5.9). Other parameters were identical to those used in Sweep 1E5–1, as described in Table 5.4.

Parameter	Start	End	Step Size
Dim_{emk}	21000	49000	14000
m	2	10	2
K	50	250	50

Table 5.7: Ranges for sLDA Sweep 1E5–2

$$e = \frac{\text{Dim}_{emk}}{mk} \tag{5.10}$$

The measured effects of m and k on $\hat{L}$ and t are again summarized in Figure 5.9. In this case, values of e are implicit and can be determined from Equation 5.10. As with Sweep 1E5–1, we again quantify Err_{emk} in Figure 5.10 and $3\sigma_{emk}$ in Figure 5.11 to confirm that the measured values are representative in our dataset. You will see that σ_{emk} and Err_{emk} take similar values to Sweep 1E5–1, with the exception of some anomalously high values for σ^t_{emk}. However, since high precision time estimation is not the focus of this study, we focused primarily on Figure 5.9. Here we can observe

the obvious anticorrelation between $\hat{L}_{emk}$ and k. However, the other anticorrelations with e, m, and Dim_{emk} from Sweep 1E5–1 are all less apparent. This could perhaps be explained by the large difference in scale between k and the other parameters.

Before investigating further, we confirmed that our observation from Section 5.1.1, that $\hat{L}_{emk}$ and Dim_{emk} are anticorrelated, still held for Sweep 1E5-2. Figure 5.12 illustrates that this still appears to be the case. As dimensionality of sLDA increases, the mean log loss of its predictions can be observed to steadily decrease. We quantify this relationship in Table 5.8 by calculating Pearson's r correlation coefficient between Dim_{emk} and $\hat{L}_{emk}$. The resulting value of $\rho_{\text{Dim},\hat{L}} = -0.178$ tells us that the anticorrelation is noisy, but measurable. Although it is difficult to accurately measure the significance of this correlation, it provides a baseline of comparison between Dim_{emk}, m, and k.

X	$\rho_{X,\hat{L}}$
m	-0.021126
k	-0.754449
Dim_{emk}	-0.178567

Table 5.8: Pearson's r Correlations of $\hat{L}_{emk}$

From here, we can quantify the qualitative trends we observed in Figure 5.9 by drawing similar plots for m and k. First, we investigate m in Figure 5.13. Here we can see that unlike Dim_{emk}, no trend can be observed between m and $\hat{L}_{emk}$. If there is any corelation between the two, it is entirely lost within the noise of the data. This differs drastically with k, as depicted in Figure 5.14. Here we can see a clear downward trend as we increase k, as well as an anticorrelation in Table 5.8 that is significantly larger than our baseline of $\rho_{\text{Dim},\hat{L}}$.

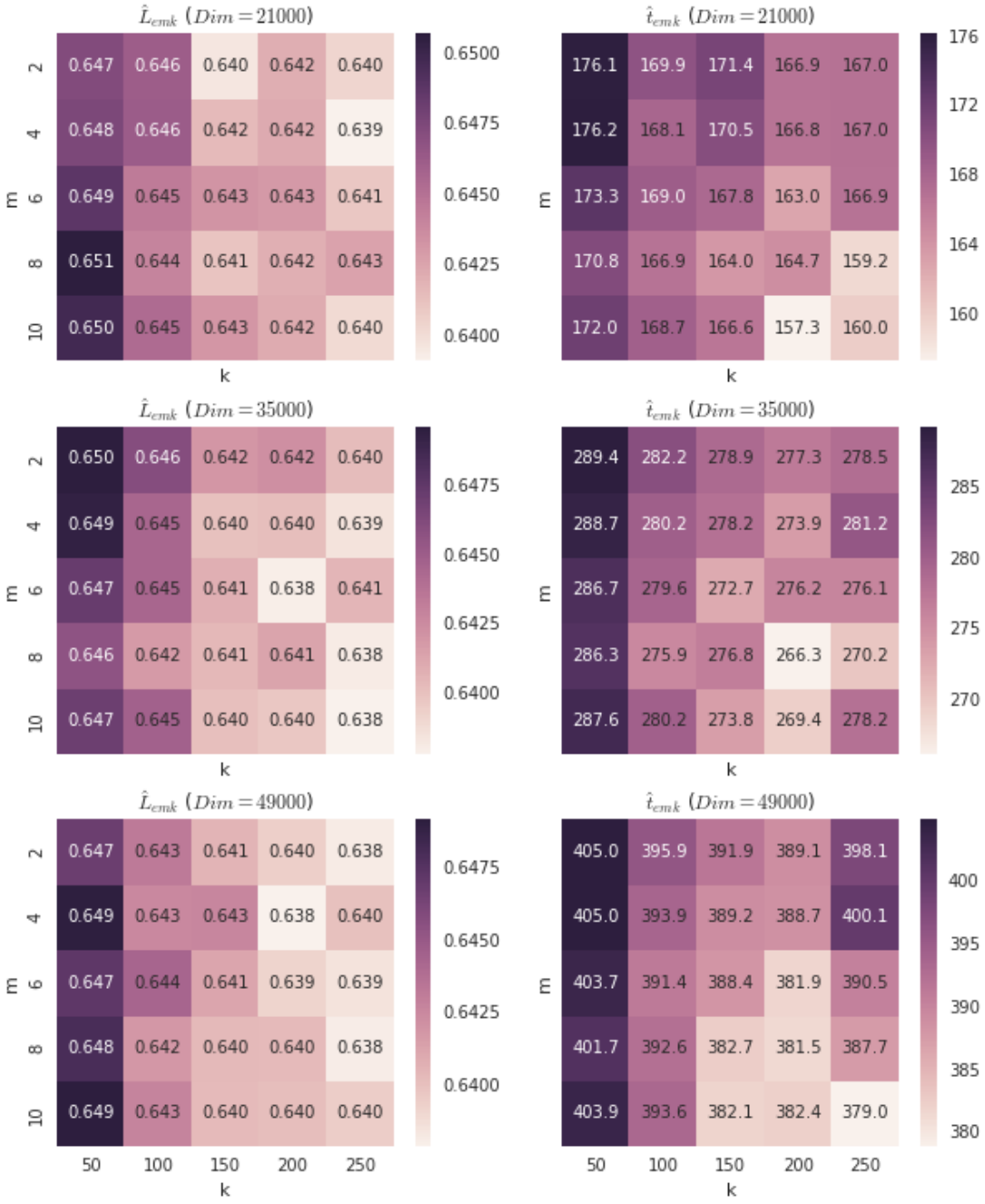

Figure 5.9: Sweep 1E5–2 Mean Log Loss and Mean Elapsed Time

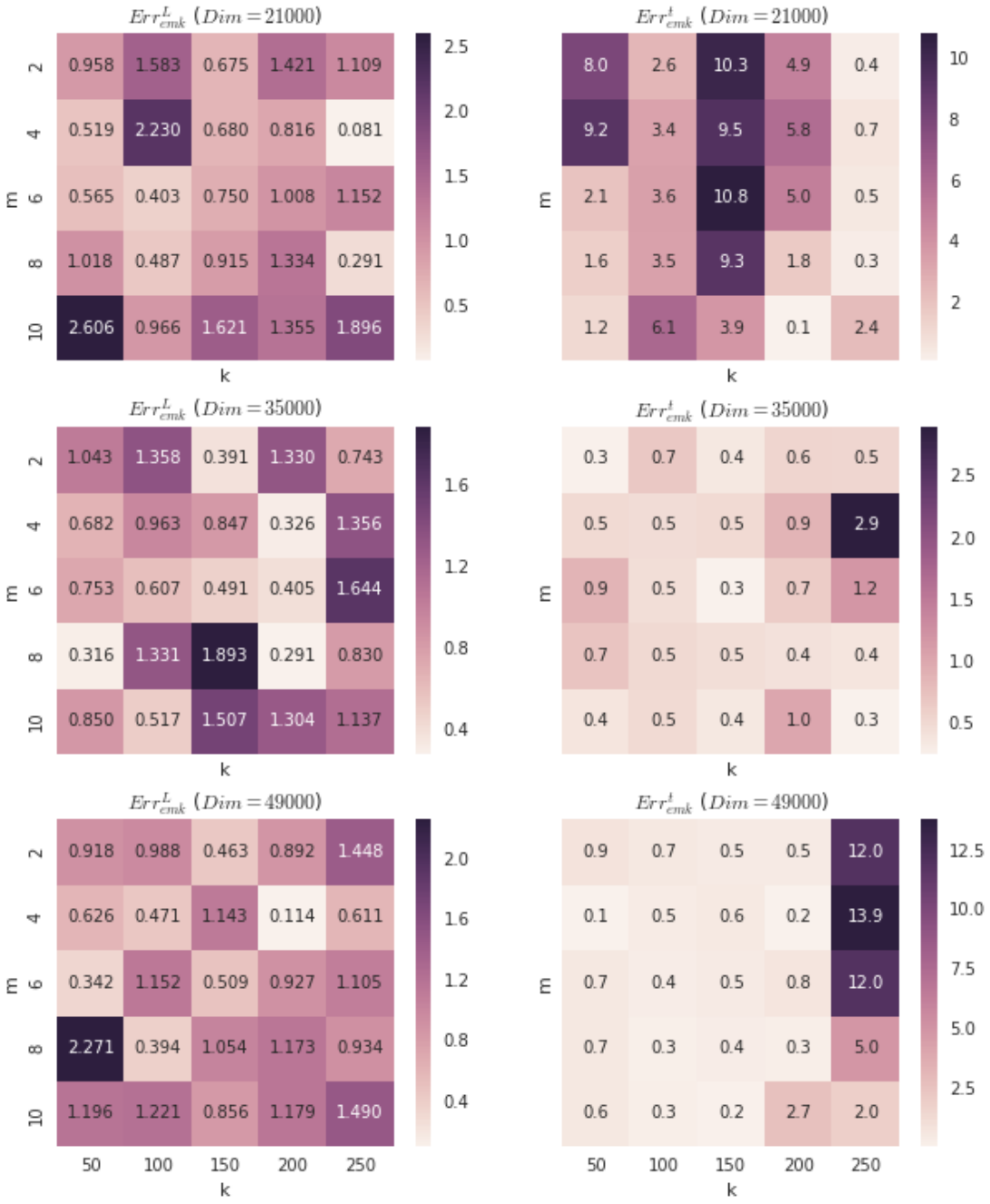

Figure 5.10: Error % of Sweep 1E5–2 Mean Log Loss and Mean Elapsed Time

Figure 5.11: $3\sigma_{emk}$ of Sweep 1E5–2 Mean Log Loss and Mean Elapsed Time

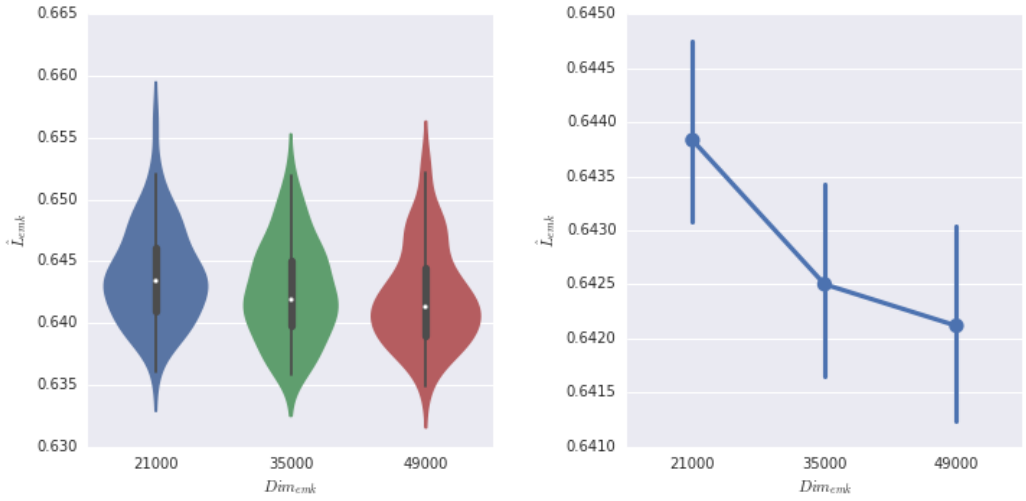

Figure 5.12: Dimensionality vs Log Loss for Sweep 1E5–2

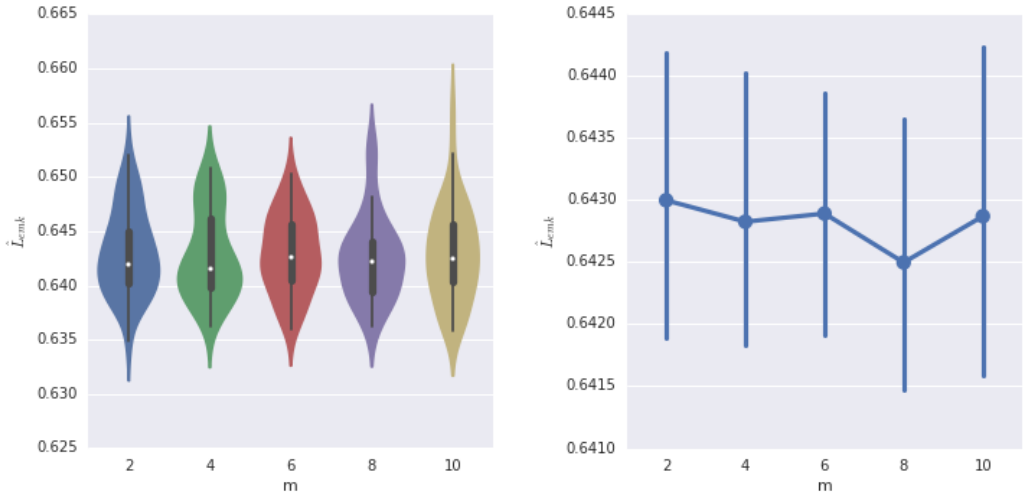

Figure 5.13: m vs Log Loss for Sweep 1E5–2

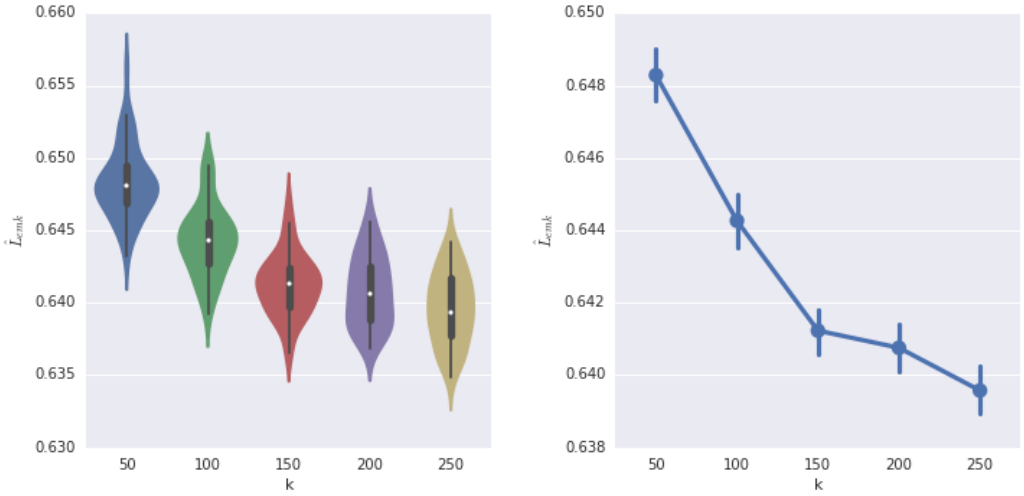

Figure 5.14: k vs Log Loss for Sweep 1E5–2

These patterns were consistent with our own understanding of the sLDA model, and the difference between e or m and k. Whereas e and m are variables that control the number of iterations given for the model to converge, k increases the model's available degrees of freedom. Therefore, after e and m are set to values sufficiently large enough for the model to converge, we would expect very little improvement in performance by increasing them further. On the other hand, increasing k allows a fitted model to capture additional information. The lack of any discernible trend between m and $\hat{L}_{emk}$ suggests that for the values considered in Sweep 1E5–2, the model has entirely converged.

5.1.3 Sweep 1E5–3

Having clearly established that k was the only variable having a measurable impact on performance, we ran Sweep 1E5–3 with the intention of exploring the limits of k's benefits in relation to the other variables. Dim_{emk} and m were set to the fixed values in Table 5.9, while k's range was increased to $[50, 1250]$ as detailed in Table 5.10. We

continued to adjust e according to Equation 5.10 as in Sweep 1E5–2, as well as 3-fold cross validation. It is important to note that in this sweep we still hold dimensionality constant, meaning that as we increase k, we have fewer iterations available in e to converge on a solution.

Parameter	Value
Dim_{emk}	35,000
m	2

Table 5.9: Static Parameters for sLDA Sweep 1E5–3

Parameter	Start	End	Step Size
K	50	1250	50

Table 5.10: Ranges for sLDA Sweep 1E5–3

From the results in Sweep 1E5–3, we found that as k increased, its benefits were eventually reversed. Figure 5.15 depicts the mean Log Loss of our sLDA model as we vary k, along with its associated error bars. We can see that the model exhibits optimal mean performance at $k = 600$, which corresponds to $e = 29$. However, it is also notable that these results are fairly noisy, and given the various local minima surrounding $k = 600$, it is possible that the true minimizing value for k might be as low as 350 or as high as 750 if we were to increase our sample size. Nonetheless, this operational range gives us enough information to estimate roughly optimal parameters for larger dataset sizes.

There are two possible explanations for the occurrence of this performance minimum. The first, and possibly more obvious explanation, was that for values of $e < 29$ the model was insufficiently converged, resulting in a performance degradation that outweighed any benefits of further increasing k. Alternatively, it could be that higher

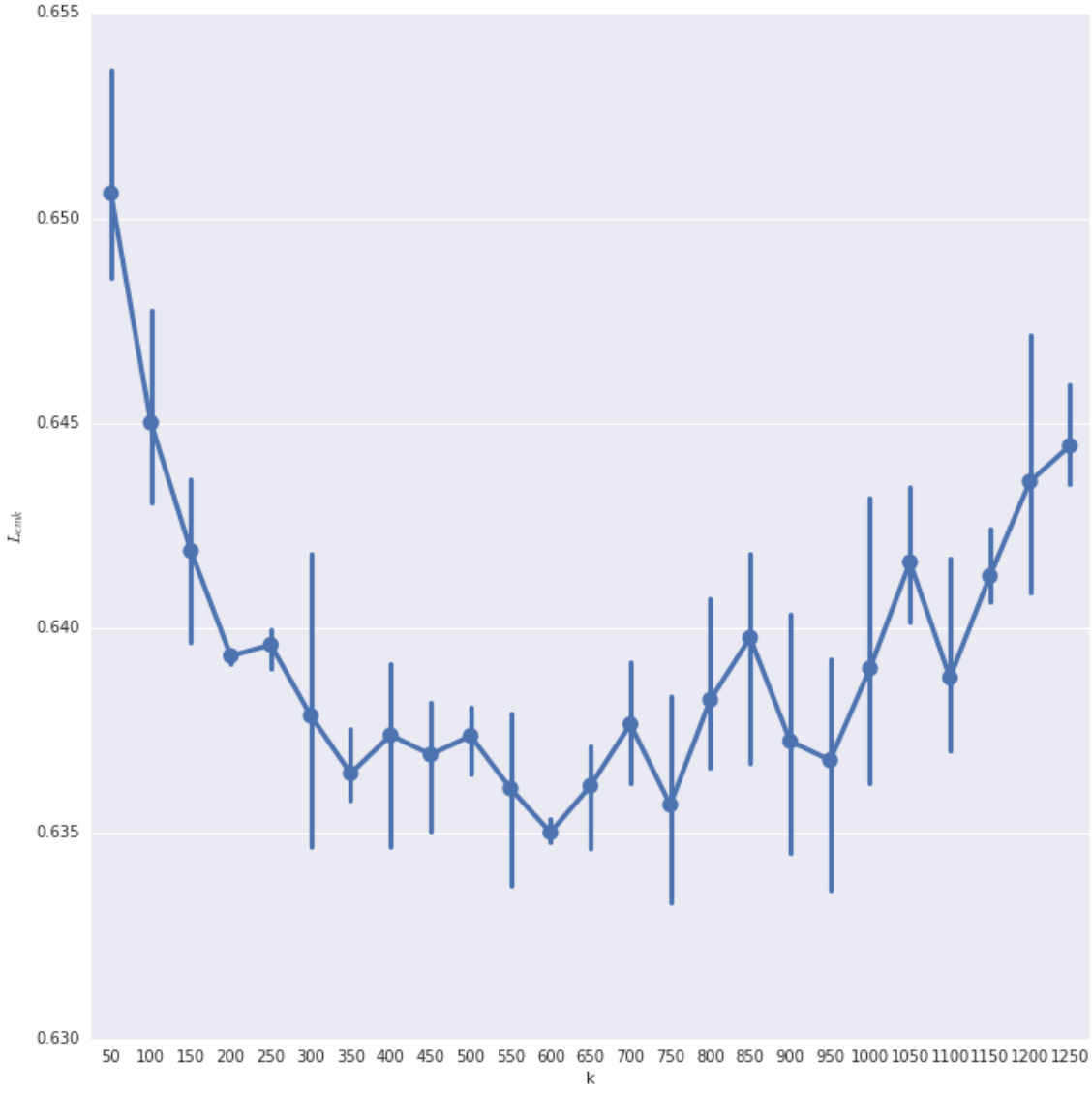

Figure 5.15: k vs Log Loss for Sweep 1E5–3

values of k did a poorer job of capturing information, contrary to our assumptions of the model. However, investigating this distinction was beyond the scope of our goals. We instead concluded that we should use $k = 600$ when training our final model for use on the test set, and moved on to measuring the performance of sLDA on the held-out testing set.

5.2 SLDA Performance Analysis

Having determined good estimates for optimal parameters for sLDA, we moved on to testing it against the held-out test set, and comparing its performance to a number of other benchmarks. We proceeded to train our sLDA model on the full TClean training set, alongside our reference models, and then test them on the corresponding held-out test set. We found that although sLDA succeeded at predicting retweets to some degree, it was outperformed by another widely available model.

In addition to sLDA, we trained the two Naive Bayes models shown in Table 5.11, as well as an ensemble classifier to test the possibility of combining the output of sLDA with that of our best performing classifier. For the sLDA model, we chose tuning parameters of $e = 200$, $m = 3$, and $k = 600$. Here we used the optimal k ascertained in Section 5.1, but chose to increase our e and m values in order to avoid the accuracy tradeoff discussed at the end of the same section. We also evaluated a simple baseline, where every prediction y was equal to the retweet ratio in the training set.

While the sLDA model continued to use the "lda" R package, the rest of the models were trained and tested using the "scikit-learn" python module. For consistency, we exported sLDA predictions to csv, and then compared them in scikit-learn using the

Model	X	Parameters
sLDA	Token Counts	$e = 200, m = 3, k = 600$
Multinomial Naive Bayes	Token TFIDF	
Bernoulli Naive Bayes	Hashtag Count	
Ensemble SGD Logistic Classifier	sLDA & Naive Bayes	loss = Log-Loss

Table 5.11: Tested Predictive Models

same methods as the other models. Table 5.12 shows a performance summary of the evaluated models, while Figures 5.16 through 5.19 show the corresponding ROC curves. SLDA performed better than the simple baseline and the hashtag-based Bernoulli Naive-Bayes, but was outperformed by the text-based Multinomial Naive-Bayes. We then took the output of sLDA and Multinomial Naive Bayes and used them as inputs to an Ensemble SGD Logistic Classifier to test the hypothesis that by combining both approaches, we could outperform Multinomial Naive Bayes alone. However, we found that the ensemble classifier had significantly worse log-loss than Multinomial Naive Bayes, indicating that the noise introduced by considering both outputs outweighed the gain in information.

Model	$\mathcal{L}$	$\mathcal{A}$
sLDA	0.611700	0.683622
Simple Baseline	0.665830	0.500000
Multinomial Naive Bayes	0.563959	0.741970
Bernoulli Naive Bayes	0.628331	0.607087
Ensemble SGD Logistic Classifier	0.595622	0.704890

Table 5.12: Summary of Predictor Performance

Although sLDA was outperformed by Multinomial Naive Bayes, we continued to investigate sLDA for its descriptive properties. Table 5.13 shows the 5 most popular topics from our sLDA model, as measured by their estimated η_k. This provides an

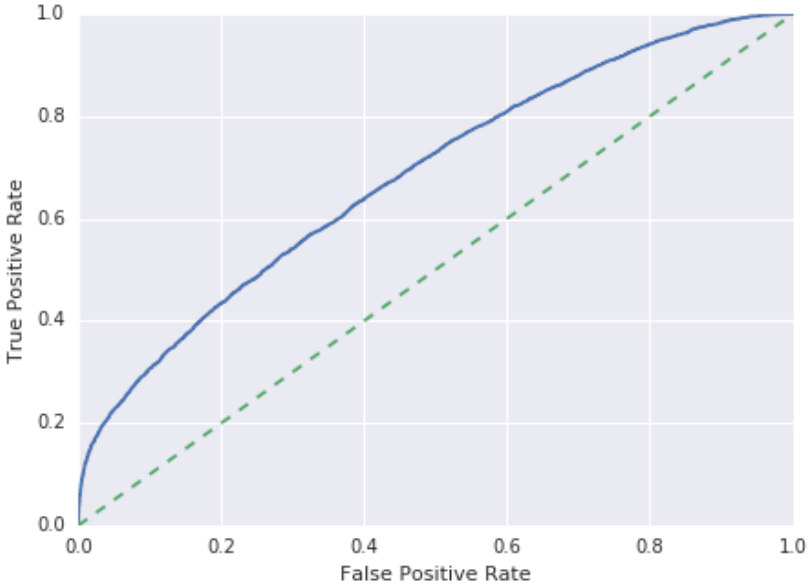

Figure 5.16: sLDA Classification ROC Curve

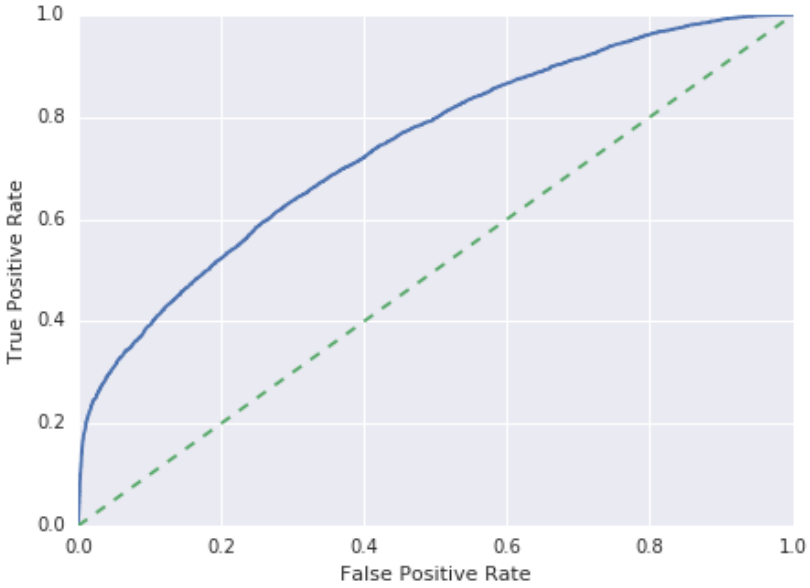

Figure 5.17: Multinomial Naive Bayes Classification ROC Curve

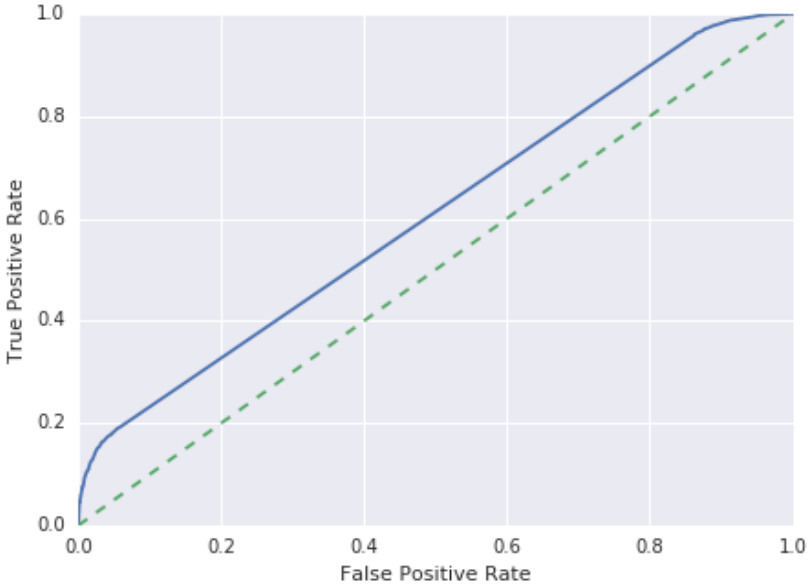

Figure 5.18: Bernoulli Naive Bayes Classification ROC Curve

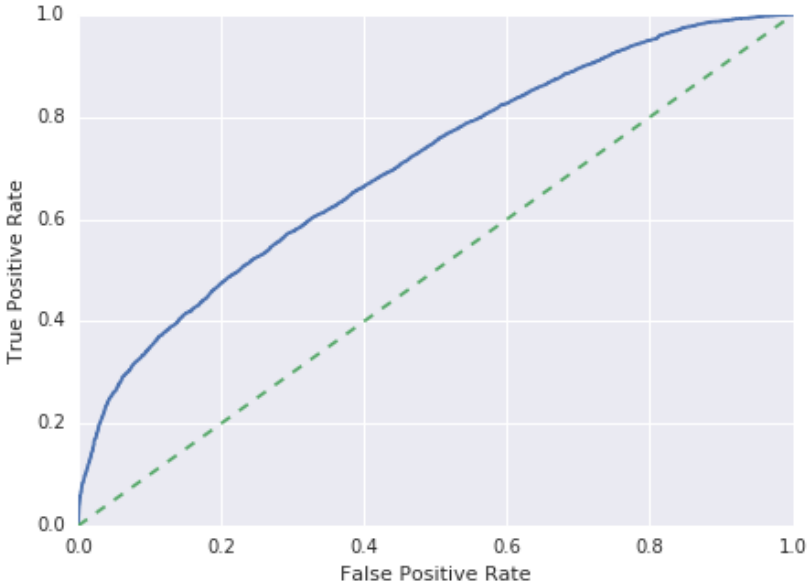

Figure 5.19: Ensemble Logistic Classification ROC Curve

insight into popularity which is more nuanced than a simple Naive Bayes approach. Qualitatively, we can see themes of fashion, romance, and money in the first three topics listed. This could easily inform potential content creators. This white-box approach of giving qualitative descriptions of popularity is a key benefit to sLDA over other algorithms, which in some cases may make it more desirable than higher-performing algorithms.

Topic k	328	44	221	147	347
η_k	1.326110	1.258460	1.210058	1.139367	1.137893
Word 1	httpt	girl		befor	thi
Word 2	look	gui	work	iv	see
Word 3	jean	hi	stai	heard	take
Word 4	#mycalvins	ar	art	thi	princess
Word 5	model	love	lil	seen	good
Word 6	underwear	thei	bank	thing	coupl
Word 7	morn	boyfriend	tip	sai	hope
Word 8	ar	human	awai	ur	week

Table 5.13: Popular Topics and Their Most Frequently Assigned Words

We concluded that although sLDA is a sub-optimal algorithm for general content-based popularity prediction on twitter, it remained useful for its capacity to qualitatively describe popular topics. We conjecture that in some cases, application developers may prefer it for this capability, despite the availability of higher performing alternatives.

CHAPTER 6

CONCLUSION

Popularity prediction on Twitter has received wide attention in the academic community. Of particular interest is content-based prediction, which despite its intrinsic challenges on short messages provides many actionable insights for content creators. In this study, we have detailed the role topic models can play in content-based popularity prediction on Twitter. We found that topic models are indeed capable of predicting retweets on Twitter, but they are outperformed by more established methods such as Naive Bayes classification.

In Chapter 3, we detailed our methods for collecting and storing 2.5T of Twitter data, gathered over the course of five months. As our study progressed, we developed more sophisticated data processing techniques in order to reconcile its scale with our analysis tools. We found that more sophisticated algorithms were generally designed for significantly smaller datasets, and adapted to this by developing the TClean dataset to accomodate more aggressive truncation. It is our hope that the datasets and collection tools from this chapter will continue to serve subsequent studies into microblog dynamics.

In Chapter 4, we showed prediction techniques using hashtags could be employed to predict retweets, consistent with previous works that identified them as strong predictive factors. We then demonstrated a technique for extending their predictive

capacity to untagged messages by correlating hashtags with their vocabulary. We also discussed some of the technical challenges of applying the relevant LLDA topic model to perform this mapping, and presented a scalable TFIDF-based alternative.

Finally, we explored the efficacy of supervised topic models for popularity prediction in Chapter 5. We presented our techniques for tuning sLDA's configuration parameters and identified the parameters that would yield optimal performance for our scenario. We then measured this performance and compared it to benchmark algorithms such as Naive Bayes and logistic regression. We found that although sLDA could make some retweet predictions successfully, it was outperformed by these more established baselines.

In summary, we found that while it was possible to predict retweet popularity based solely on a tweet's content, topic models were not the best tool for the job. More established methods such as Naive Bayes were more effective for popularity prediction tasks. Instead, the utility of topic models may be seen in their capacity to identify popular topics of discussion, and therefore provide a more transparent description of tweet popularity.

6.1 Future Works

The research conducted here could provide the foundation for many future works. Among them are:

- More sophisticated optimization of sLDA parameter searches. Although the iterative sweeps in Chapter 5 gave us sufficient information to continue with our experimentation, potential performance improvements could be realized by searching for optimal parameters via non-linear programming algorithms.

- Large-scale implementations of prediction algorithms. With the scale at which messages occur in social media, any candidate prediction algorithm needs to be able to analyze messages at scale for it to be used in production. While our study focused on performance over scalability, there is a growing need for any candidate algorithm to scale well.

- Time-aware analysis. The current training regime considers a dataset where the test set is sampled randomly from within the corpus. It would be an interesting point of study to see how well a model's predictive performance varied across time. The length of our TClean corpus would lend itself well to this area of investigation.

- Leveraging qualititative descriptions from topic models. Descriptions of popular topics by sLDA are perhaps its most distinguishing features from other models. If time permitted, we would have liked to explore different applications that made use of this feature.

REFERENCES

[1] Cascading — cascading. http://www.cascading.org/projects/cascading/. Accessed: 2015-11-05.

[2] Get statuses/sample — twitter developers. https://dev.twitter.com/streaming/reference/get/statuses/sample. Accessed: 2015-11-04.

[3] myleott/jgibblabeledlda. https://github.com/myleott/JGibbLabeledLDA. Accessed: 2015-12-02.

[4] Rest apis — twitter developers. https://dev.twitter.com/rest/public. Accessed: 2015-11-04.

[5] The streaming apis — twitter developers. https://dev.twitter.com/streaming/overview. Accessed: 2015-11-04.

[6] Twitter usage statistics - internet live stats. http://www.internetlivestats.com/twitter-statistics/. Accessed: 2015-11-04.

[7] Eytan Bakshy, Jake M Hofman, Winter A Mason, and Duncan J Watts. Everyone's an influencer: quantifying influence on twitter. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 65–74. ACM, 2011.

[8] Peng Bao, Hua-Wei Shen, Junming Huang, and Xue-Qi Cheng. Popularity prediction in microblogging network: a case study on sina weibo. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 177–178. International World Wide Web Conferences Steering Committee, 2013.

[9] Bin Bi, Yuanyuan Tian, Yannis Sismanis, Andrey Balmin, and Junghoo Cho. Scalable topic-specific influence analysis on microblogs. In *Proceedings of the 7th ACM international conference on Web search and data mining - WSDM '14*, pages 513–522, New York, New York, USA, 2014. ACM Press.

[10] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, March 2003.

[11] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and P Krishna Gummadi. Measuring user influence in twitter: The million follower fallacy. *ICWSM*, 10(10-17):30, 2010.

[12] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and P Krishna Gummadi. Measuring user influence in twitter: The million follower fallacy. *ICWSM*, 10(10-17):30, 2010.

[13] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[14] Shuai Gao, Jun Ma, and Zhumin Chen. Effective and effortless features for popularity prediction in microblogging network. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 269–270, Republic and Canton of Geneva, Switzerland, 2014. International World Wide Web Conferences Steering Committee.

[15] Shuai Gao, Jun Ma, and Zhumin Chen. Modeling and predicting retweeting dynamics on microblogging platforms. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 107–116. ACM, 2015.

[16] Liangjie Hong, Ovidiu Dan, and Brian D. Davison. Predicting popular messages in twitter. In *Proceedings of the 20th International Conference Companion on World Wide Web*, WWW '11, pages 57–58, New York, NY, USA, 2011. ACM.

[17] Zhiting Hu, Junjie Yao, Bin Cui, and Eric P Xing. Community Level Diffusion Extraction. In *Sigmod'15*, pages 1555–1569, New York, New York, USA, May 2015. ACM Press.

[18] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 56–65. ACM, 2007.

[19] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*, page 137, New York, New York, USA, August 2003. ACM Press.

[20] Dominic L Lasorsa, Seth C Lewis, and Avery E Holton. Normalizing twitter: Journalism practice in an emerging communication space. *Journalism studies*, 13(1):19–36, 2012.

[21] Zongyang Ma, Aixin Sun, and Gao Cong. On predicting the popularity of newly emerging hashtags in twitter. *Journal of the American Society for Information Science and Technology*, 64(7):1399–1410, 2013.

[22] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[23] Jon D Mcauliffe and David M Blei. Supervised topic models. In *Advances in neural information processing systems*, pages 121–128, 2008.

[24] Rishabh Mehrotra, Scott Sanner, Wray Buntine, and Lexing Xie. Improving lda topic models for microblogs via tweet pooling and automatic labeling. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 889–892, New York, NY, USA, 2013. ACM.

[25] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] Adam L Penenberg. *Viral loop: from Facebook to Twitter, how today's smartest businesses grow themselves*. Hachette Books, 2009.

[27] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 569–577. ACM, 2008.

[28] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[29] Daniel Ramage, ST Dumais, and DJ Liebling. Characterizing Microblogs with Topic Models. In *ICWSM*, 2010.

[30] Daniel Ramage, Susan Dumais, and Dan Liebling. Characterizing Microblogs with Topic Models. In *ICWSM*, 2010.

[31] Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D Manning. Labeled LDA : A supervised topic model for credit attribution in multi-labeled corpora. In *Conference on Empirical Methods in Natural Language Processing*, number August, pages 248–256, 2009.

[32] Manuel Gomez Rodriguez, Jure Leskovec, and Bernhard Schölkopf. Modeling information propagation with survival theory. *arXiv preprint arXiv:1305.3616*, 2013.

[33] B. Suh, Lichan Hong, P. Pirolli, and Ed H. Chi. Want to be retweeted? large scale analytics on factors impacting retweet in twitter network. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 177–184, Aug 2010.

[34] Li Wei and Andrew McCallum. Pachinko allocation: DAG-structured mixture models of topic correlations. *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 577–584, 2006.

[35] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. Twitterrank: Finding topic-sensitive influential twitterers. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 261–270, New York, NY, USA, 2010. ACM.

[36] Lei Yang, Tao Sun, Ming Zhang, and Qiaozhu Mei. We know what@ you# tag: does the dual role affect hashtag adoption? In *Proceedings of the 21st international conference on World Wide Web*, pages 261–270. ACM, 2012.

[37] Jiawei Zhang. University of Illinois at Chicago Phd Qualifier Examination Paper Link Prediction across Heterogeneous Social Networks : A Survey. 2014.

[38] WayneXin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, and Xiaoming Li. Comparing twitter and traditional media using topic models. In Paul Clough, Colum Foley, Cathal Gurrin, GarethJ.F. Jones, Wessel Kraaij, Hyowon Lee, and Vanessa Mudoch, editors, *Advances in Information Retrieval*, volume 6611 of *Lecture Notes in Computer Science*, pages 338–349. Springer Berlin Heidelberg, 2011.

APPENDIX A

COMPUTING RESOURCES

A.1 BDServer Hadoop Cluster

BDServer refers to the primary BSU Hadoop cluster in use during the majority of this research. It was used to store our Twitter datasets, as well as perform MapReduce processing tasks upon them. It consisted of 6 homogeneous machines with the specifications described in Table A.1. To improve MapReduce performance, they were internally networked on an InfiniBand interconnect.

CPU Model	Intel®Xeon®CPU E5-1410
CPU Clock Speed	2.80 GHz
CPU Cache Size	10240 KB
Physical Memory Capacity	16 GB
Physical Memory Type	DDR3
HDD System Storage	500 GB
HDD Hadoop Storage	6 TB
SSD Hadoop Storage	128 GB
Operating System	CentOS release 6.5 (Final)

Table A.1: BDServer Node System Specifications

A.2 Infolab

The Infolab server was a BSU server with large memory capacity. It was used in the experiments from Chapter 4 for processes that had high memory requirements. Its

specifications are shown in Table A.2

CPU Model	AMD Opteron™Processor 6320
CPU Clock Speed	2.80 GHz
CPU Cache Size	2048 KB
Physical Memory Capacity	128 GB
Physical Memory Type	DDR3
HDD System Storage	1 TB
HDD Data Storage	1 TB
Operating System	Ubuntu 14.04.3 LTS

Table A.2: Infolab System Specifications

A.3 Sweet Chedda

Sweet Chedda was a home PC that was used for running a number of experiments. Its specifications are shown in Table A.3.

CPU Model	Intel®Core™i7-4770K CPU
CPU Clock Speed	3.50 GHz
CPU Cache Size	8192 KB
Physical Memory Capacity	16 GB
Physical Memory Type	DDR3
HDD System Storage	2 TB
SSD System Storage	500 GB
Operating System	Arch Linux

Table A.3: SweetChedda System Specifications