Honours Year Project Report

Predicting Web 2.0 Thread Updates

By

Shawn Tan

Department of Computer Science

School of Computing

National University of Singapore

2012

Honours Year Project Report

Predicting Web 2.0 Thread Updates

By

Shawn Tan

Department of Computer Science

School of Computing

National University of Singapore

2012

Project No: H079830
Advisor: A/P Kan Min-Yen
Deliverables:
Report: 1 Volume

Abstract

With the advent of Web 2.0, sites with forums, or similar thread-based discussion features are increasingly common. An incremental web crawler aiming to maintain a database of up-to-date, extracted information from sites with such discussion features must strike a balance between bandwidth usage and freshness of data. Our objective: To estimate the arrival times of the next update to such threads. We demonstrate three different methods for achieving this using regression methods, and make recommendations as to how they can be used in a crawling system. We also propose a novel metric for measuring the timeliness of such a model that balances between the model's timeliness and bandwidth consumption. Our methods outperform the baseline, which revisits the thread using its average update rate.

Subject Descriptors:
World Wide Web
Web searching and information discovery
Web search engines
Web crawling
Information Retrieval
Evaluation of retrieval results
Retrieval effectiveness

Keywords:
web crawlers, revisitation, discussion threads, evaluation metrics

Implementation Software and Hardware:
python, bash, scikit-learn

Acknowledgement

I would like to express my gratitude to my parents, for supporting me throughout my final year of university. I would also like to thank the following friends: Low Wee, Yipeng, Cedric, Davin, Chris and Lan Guan. Without them, I would not have had the moral support during times of stress, nor would I have been able to have discussions while fleshing out ideas.

Most of all, I would really like to thank Jesse and A/P Min-Yen Kan, for their patient guidance, support, and for putting up with my stubbornness and tardiness during the course of this project.

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

With the advent of Web 2.0, sites with forums, or similar thread-based discussion features are increasingly common. Table 1.1 shows us that many of the popular Web 2.0 sites have comment features. This suggests that content on the web is increasingly being created by users alongside content providers. Our goal in this thesis is to create an algorithm that can predict when updates in such discussion threads will occur.

A naive way of getting timely updates would be to aggressively hit the pages repeatedly (polling), downloading pages at a frequent rate. Since web crawling is largely IO-bound, a large portion of the time spent crawling would be spent waiting for the server to supply a response to the request issued by the crawler. However, sites with a large number of pages (like popular forum sites), makes this infeasible in practice. On top of the usual requests the server may have, it then has to deal with repeated requests from such a crawler. Most sites do not mind some additional bandwidth, but if it gets excessive, it may be construed as a Denial-of-Service attack. These sites may then deny any further requests from the crawler.

A simple method to reduce the amount of polling done is to use the average time differences between all of the previous page updates to estimate the arrival of the next update, and to abstain from polling until the estimated time.

	T	FB L	FB S	G +1	L	DL	C	PV	Follows
http://www.lifehacker.com	1	1	1				1	1	
http://digg.com/	1	1			1	1	1	1	
http://9gag.com/	1	1	1	1	1		1	1	
http://www.flickr.com/					1		1	1	
http://news.ycombinator.com/					1		1		
http://stackoverflow.com/					1		1	1	
http://www.youtube.com/					1	1	1	1	
http://www.reddit.com/					1	1	1		
http://www.stumbleupon.com/					1		1	1	
http://delicious.com/	1	1					1	1	1

Table 1.1: Features of popular Web 2.0 sites

T = Twitter mentions

FB L = Facebook Likes

FB S = Facebook Shares

G +1 = Google +1

L = Likes (Local)

DL = Dislikes (Local)

C = Comments

PV = Page Views

Follows = Site-local feature for keeping track of user's activities

A key observation in our work is that the contents of the thread may also influence the discussion and hence the rate of commenting. For example, a thread in a technical forum about a Linux distribution may start out as a question. Subsequent questions that attempt to either clarify or expand on the original question may then be posted, resulting in a quick flurry of messages. Eventually, a more technically savvy user of the forum may come up with a solution, and the thread may eventually slow down after a series of messages thanking the problem solver. We believe that the content of the thread has information that can give a better estimate of the time interval between the last post and a new one.

Let us define all such thread-based discussion styled sites as forums. Ideally, an incremental crawler of such user-generated content should be able to maintain a fresh and complete database of content of the forum that it is monitoring. However, doing so with the previously mentioned naive method would, incur excessive costs when downloading un-updated pages, and raise the possibility of the web master blocking the requester's IP address.

As such, we need to have a way to achieve a balance between two things: (1) Reduce the requests made, and (2), still be able to retrieve updates in a timely fashion. There has been work done (see Chapter 2) in this respect, but we argue that existing content should be included into any model that attempts to predict updates to user-generated content.

Our high level goal: To devise a suitable algorithm for predicting new posts in user discussion threads, based on the discussion content in the thread. In this project, we focus on forum threads, and make the following contributions:

1. We demonstrate three different methods for achieving this using regression methods.

2. We propose a novel metric for measuring the timeliness of such a model that balances between the model's timeliness and bandwidth consumption.

In Chapter 2, we explore the related work dealing with refresh policiesand metrics to measure the performance of such algorithms. In Chapter 3, we discuss the methods that we have created with to tackle the problem, while Chapter 4 describes the metrics we propose for measuring the performance of these algorithms. In Chapter 5, we perform experiments on a dataset extracted from `avsforum.com`, and show that our models perform better than an average revisitation baseline. Chapter 6 then discusses our contributions, and possible avenues of future work.

Chapter 2

Related Work

In this chapter, we present some of the work done related to prediction of thread updates. First, we take a look at the literature dealing with incremental crawlers and their policies for revisiting a web page. Some work has been done to try to predict how often page content is updated, with the aim of scheduling download times in order to keep a local database fresh. We then also take a brief look at some work dealing with content. We also review some work related to evaluation metrics.

2.1 Refresh policies for incremental crawlers

We first discuss the *timeliness* of our crawler to maintain the freshness of the local database, which refers to how new the extracted information is. Web crawlers can be used to crawl sites for user comments for later post-processing. Web crawlers which maintain the freshness of a database of crawled content are known as incremental crawlers. Two trade-offs these crawlers face cited by Yang, Cai, Wang, and Huang (2009) are *completeness* and *timeliness*. *Completeness* refers to the extent which the crawler fetches all the pages, without missing any pages. *Timeliness* refers to the efficiency with which the crawler discovers and downloads newly-created content. We focus mainly on timeliness

in this project, as we believe that timely updates of active threads are more important than complete archival of all threads in the forum site.

Many such works have used the Poisson distribution to model page updates. This is because the Poisson process is often used to model a sequence of events that happen independently, with an average rate, over time. Coffman and Liu (1997) analysed the theoretical aspects of doing this, as well as formalised the problem. Their work showed that if the page change process is governed by a Poisson process with a rate of μ, then accessing the page at intervals proportional to μ is optimal.

Cho and Garcia-Molina trace the change history of 720,000 web pages collected over four months, and showed empirically that the Poisson process model closely matches the update processes found in web pages (Cho, 1999). They then proposed different revisiting or refresh policies (Cho & Garcia-Molina, 2003; Cho & Garcia-molina, 2003) that attempt to maintain the freshness of the database.

The Poisson distribution were also used in Tan, Zhuang, and Mitra (2007), where they described a method that made use of a weighted history, such that recent changes are more important than older ones.

However, the Poisson distribution is memoryless, and in experimental results due to Brewington and Cybenko (2000), the behaviour of site updates are not. Moreover, these studies were not performed specifically on online threads, where the behaviour of page updates differs from static pages.

Yang et al. (2009), attempted to resolve this by using the list structure of forum sites to infer a sitemap. With this, they reconstruct the full thread, and then use a linear-regression model to estimate a scoring function. This function is then used to schedule when the next visit to the thread will be made.

Forums have a logical, hierarchical structure in their layout, which typically alerts the user to thread updates by putting threads with new replies at the top of the thread index.

Yang's work exploits this as well as their linear model to achieve a prediction of when to retrieve the pages. However this design pattern is not applied universally; comments on blog sites or e-commerce sites about products do not conform to this pattern. The lack of such information may result in a poorer estimate, or no estimate at all.

What we are trying to do here is similar. We also want to create a revisit policy, but we argue that the previous rates of change should not be the only factor that is taken into account when coming up with such a policy. The above works all try to estimate the arrival of the next update (comment), but do not leverage an obvious source of information, which is the content of the posts themselves. Our perspective is that the available thread content can be used to provide a better estimation for predicting page updates.

Next, we look at some of the related work pertaining to thread content.

2.2 Thread content analysis

While there is little existing work using content to predict page updates, we review existing work related to analysing thread-based pages. We think such work will aid our efforts in content-based update prediction.

Wang and McCarthy (2011) find hierarchical links between forum posts using lexical chaining. They proposed a method to link posts using the tokens in the posts called $Chainer_{SV}$. While they analyse the contents of individual posts, the paper does not make any prediction with regards to newer posts.

There has also been some work done recently in predicting events in social media, and in particular, tweets. Wang, Chen, and Kan (2012) dealt with predicting the retweet-ability of tweets using content. They applied two levels of classification, the first level categorising tweets into 6 different types: Opinion, Update, Interaction, Fact, Deals and Others. This was done using similar techniques as Sriram and Fuhry (2010) and Naaman,

Boase, and Lai (2010). The Opinion and Update categories are then further categorised into another three and two sub-categories each. The authors performed this categorisation using labeled Latent Dirichlet Allocation. These classifications are then used as features to predict the retweetability of a tweet.

While this work leverages content to make predictions about user responses, the predictions are made into three classes: no retweets, low retweets, and high retweets. While this may not be directly useful, some of the features used in their work could be leveraged to make a more accurate prediction.

2.3 Evaluation metrics

Yang et al. (2009) proposed a metric for our particular problem of thread update prediction. Known as the T-score, it gives the average time difference between when a post is made and when the post is retrieved by a crawler. The lower the T-score, the better the model. However, the metric does not penalize for visits which retrieve nothing new from the thread. As such, a crawler that repeatedly crawls the site at a frequent rate would do very well.

Broadening our search for more relevant evaluation metrics that take such wasted bandwidth into account, we turn to related work in the evaluation of segmentation algorithms. In Georgescul, Clark, and Armstrong (2009), the authors propose a new scheme for evaluating segmentation algorithms, Pr_{error}. This metric is the weighted sum of two probability counts Pr_{fa} which is the probability that a false alarm segmentation is made, and Pr_{miss} which is the probability that a segmentation is not made when there should be one. Unfortunately for our purposes, the metrics are calculated using the number of ground truths and segmentations given a window. As such, it does not account for the "distance" between the ground truths and the segmentation. It also does not allow for the predictions to appear after the ground truths, all requirements needed for a metric

to evaluate timeliness of a model.

The metric proposed here, however do not penalise segmentations that come before the ground truths. We build on the same ideas to create a metric for evaluating our methods (See Chapter 4).

2.4 Conclusion

The state of current work related to revisitation policies mainly use estimations of previous update intervals to predict future update times. Analysis of user-generated content also do not tackle the problem of predicting when new content is created or published. These are the issues we will tackle with our work.

We aim to use the existing content available in the thread to train models for predicting when future posts will arrive. In the next chapter, we take a look at the various methods we propose for tackling this problem of revisitation.

Chapter 3

Method

We aim to predict the amount of time between the arrival of the next upcoming post and the time the last post in the thread was made. The information available to us are the previous posts observed in prior visits. The assumption made here is that the thread is not paginated in any way, and a single visit to the thread gives us the latest posts without having to traverse through the links to the latest page. This is because in practice, we would be able to keep track of where the last visited page of the thread was, and reading the new posts would incur a few more requests to the thread. This, in comparison to constantly hitting the page for updates, would be negligible.

More formally, what we are trying to do is to estimate a function f such that given a feature vector $\mathbf{X}$ representative of a window $\rho_{t-w+1}, \rho_{t-w+2}, \cdots, \rho_t$, where ρ_t represents the t-th post in the thread, we can approximate Δ_t with $f(\mathbf{X})$. In the following sections, we will discuss various methods for estimating f, following the notation introduced in Table 3.1.

We now describe the methods we explored to approximate f.

Notation	Description
ρ	A post
t	Index of a post in a thread
w	Number of posts in a window
ρ_t	The t-th post in the thread
$\mathbf{v}_t$	The frequency count vector of the posts used in the t-th post
Δ_t	Time difference between a post at position t and a post at position $t+1$
$\mathbf{t}_\Delta$	Vector of Δ_ts in a given window
$\mathbf{t}_{\text{ctx}}$	Bit vector representing the day of week, and the hour of day
$\mathbf{X}$	Feature vector extracted from a window
K	The K best features selected from the vocabulary.

Table 3.1: Common notation used throughout this thesis

3.1 Baselines

A simple way of estimating the revisit rate is to use the average time difference given past observed posts. In our review of the related work, we have seen that if page updates follow a Poisson distribution, then revisiting at the Poisson mean would be an optimal approximation (Coffman & Liu, 1997). In our baseline revisit policy, we account for the last made post whenever we make a visit to the thread, and calculate our next revisit time based on the average post intervals added to the timestamp of the previous post. This is in contrast to an even simpler revisit policy that just revisits at a constant, fixed rate, independent of the posts being made to the thread.

One other way of predicting using average post intervals would be to use the concept of a *window*. Averaging the time differences between the posts intuitively works as it captures the thread's context: A series of posts with short intervals should mean that

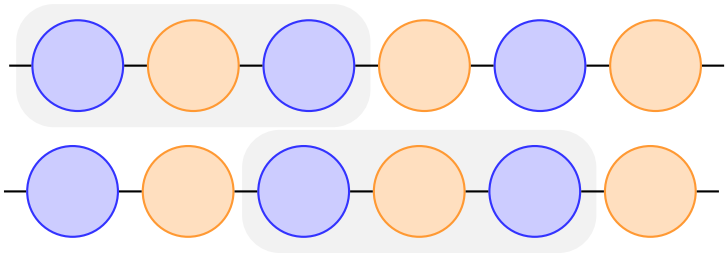


Figure 3.1: A series of events, posts (blue) and visits (orange). The diagram demonstrates the concept of a window of $w = 2$.

the next post would come at around the same interval as the few that came before.

An example of a window ($w = 2$) can be seen in Figure 3.1.

In terms of using content for prediction, windows also makes intuitive sense: Forum users view content as paginated posts, so whether or not they make a reply depends on the content they view when viewing the thread.

3.2 Performing regression on windows (SVR)

Previous work used linear regression on a set of features extracted from forums (Yang et al., 2009). In this past work, the regressed function was used as a scoring function rather than a predictive function. We implemented their model, but the results proved to perform worse than the baselines.

Informed by the past work, we use some features from the previous work: Window posts time differences and time context features (bit-vector representations of the day of the week and hour of the day).

However, this time instead of linear regression, we used a regression method known as Support Vector Regression (SVR). SVR is an extension of using Support Vector Machines for classification. Since support vector optimisation does not depend on the dimension of the input space, SVR will have advantages in high dimension feature vectors (Drucker,

Burges, Kaufman, Smola, & Vapnik, 1997). We are also using a radial basis function kernel.

The main focus of study in this report was to see if content helps with predicting thread updates would produce an improvement. In this project we used word counts. We performed standard preprocessing steps of removing stopwords and tokens that are less than three characters in length. We also stem the words using the Porter stemmer. We then perform a word frequency count. However, the use of the full vocabulary of the thread as a feature vector greatly increases the time needed to train the model. As such, we used a simple univariate regression technique for feature selection, and selected only the K best tokens for consideration. Table 5.1 shows the results of this experiment. The methods in this section use features extracted from the current window. A model is then trained using these extracted features in order to make a prediction. We take a look now at a two other novel methods that we developed.

3.3 Discounted sum of previous instances (DEC)

Posts made further in the history of the thread may have an effect on when the latest posts arrive. The magnitude of this effect, however, may diminish over time.

Instead of having a finite window for which all posts (in said window) are treated equally, we propose to accord different weights for the previous posts: the earlier the posts were made, the less effect they should have on prediction the post should have.

Following this intuition we used a discounted sum over previous posts' word frequency vector:

$$\mathbf{X}'_t = \mathbf{X}_t + \alpha \mathbf{X}'_{t-1}$$

where $\mathbf{X}_t$ is the feature vector at post t. α is the *discount factor* and satisfies $0 \leq \alpha < 1$.

This new feature vector $\mathbf{X}'_t$ will be used in the same way as before; instances of $\mathbf{X}'$ will be regressed with their Δ_t values. As before, we will look at the results for this method

in the next chapter.

Up till now, we have looked at methods that treat the model as static – once trained, the model never gets updated during run time. However, this is unrealistic due to the fact that over time, different words are popular as a direct result of different topics in the real world being popular. In this case, these fluctuations may be due to new updates to firmware being released or newer models of a product.

3.4 Stochastic Gradient Descent (SGD)

To address this problem, we investigate the use of a stochastic gradient descent (SGD) to estimate the function f. SGD allows us to use a dynamic function instead of a static function for estimation during runtime. We continue to allow f to vary whenever new posts and their update times are observed.

Having already attempted using linear regression for this purpose, we have found it unsuitable for f to be estimated by a linear function. Such a linear function often resulted in a negative prediction, and sometimes an overly huge one, when given feature vectors that have previously never been observed. The function has to be somehow constrained such that the value returned never drops below 0, and never predicts something too huge such that many posts are missed.

Since $f(\mathbf{X}) > 0$, we used a scaled sigmoid function,

$$f(\mathbf{X}) = \frac{\Lambda - \lambda}{1 + e^{\mathbf{w} \cdot \mathbf{X}}} + \lambda$$

where Λ and λ are the scaling factors. This results in $f : \mathbb{R}^{|\mathbf{X}|} \to (\lambda, \Lambda)$. Bounding the estimation function between λ and Λ allows us to restrict the prediction from becoming negative or exceedingly large. For our purposes, we set $\lambda = Q_3 + 2.5(Q_3 - Q_1)$, where Q_n is the value at the n-th quartile. A visual interpretation of such a curve can be seen in Figure 3.2.

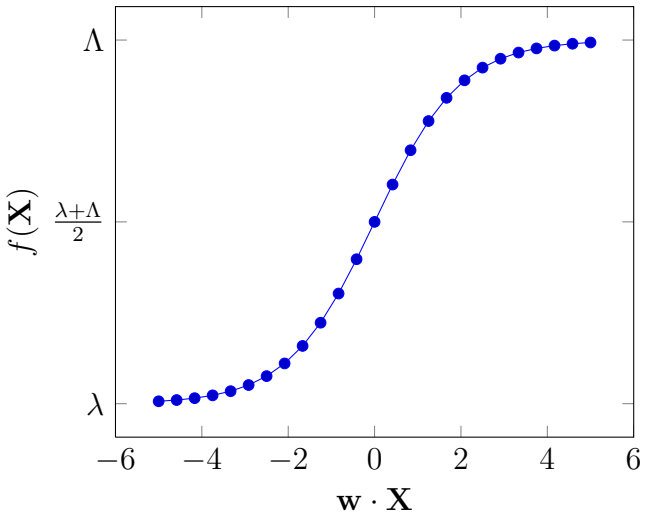


Figure 3.2: Scaled sigmoid curve

The resulting update rule for $\mathbf{w}$ is then given by,

$$\Delta \mathbf{w}_i = \eta \underbrace{\left(\widehat{\Delta_t} - \Delta_t\right)}_{\text{error term}} \underbrace{(f(\mathbf{X})(1 - f(\mathbf{X})))}_{\text{gradient}} \mathbf{X}_i$$

which is similar to the delta update rule found in artificial neural networks. We omit the scaling factor in the gradient as it is a constant and then experiment with various values of η, the learning rate.

In this chapter, we have outlined the specific task we will be attempting, to try and predict the time from the current last post in the thread to the next. We have discussed the types of features we will be using,

- $\mathbf{t}_{\text{ctx}}$: time context features
- $\mathbf{v}$: content features, in particular, word counts
- $\mathbf{t}_{\Delta}$: Δ_t of posts within the window

We have also discussed the concept of a window, and how it could help to make predictions better. The methods we will evaluate in the next chapter are following:

- `BL`: Baseline method, using the average Δ_t revisit strategy.

- `SVR`: Support vector regression, using feature vector extracted from the current window.
- `DEC`: Uses the discounted sum of previous instances as the current window.
- `SGD`: Uses the stochastic gradient descent method for prediction.

We will perform some experiments, and look at how these methods stack up against one another.

Chapter 4

Evaluation Metrics

One of the contributions of this project was also to come up with a good metric for measuring the performance of a model that performs predictions.

In trying to do this, we must first consider the two constraints that we are trying to balance: (1) The bandwidth consumption of our algorithm, and (2) the timeliness of our predictions. A violation of either should incur a penalty in the metric we are using. Such a metric could also be parameterised so that different levels of importance could be given to (1) or (2), depending on the situation.

In the next section, we will discuss two metrics that can be used to measure this. The following section then demonstrates how the two metrics can be normalised, and then combined to form our final Pr_{error} metric.

4.1 T-score, and the Visit/Post ratio

We also want to know the *timeliness* of the model's visits. Yang et al. (2009) has a metric for measuring this. Taking Δt_i as the time difference between a post i and it's download time, the timeliness of the algorithm is given by

$$T = \frac{1}{|P|} \sum_{i=1}^{|P|} \Delta t_i$$

Notation	Description
P	List of posts.
V	List of visits.
T	A thread's T-score.
$T_{\max}$	A thread's maximum T-score.
$t(\rho)$	Timestamp of the post.

Table 4.1: Notation used for evaluation metrics

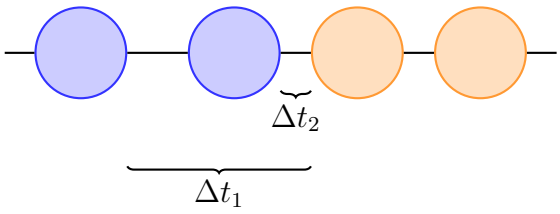


Figure 4.1: Demonstrating how T-score is measured.

A good algorithm would give a low T-score. However, a crawler that hits the site repeatedly performs well according to this metric. The authors account for this by setting a bandwidth (fixed number of pages per day) for each iteration of their testing. In our experimental results, we also take into account the number of page requests made in comparison to the number of posts.

4.2 Normalising the T-score and Visit/Post ratio

We now have two different metrics that we need to combine into a single score to measure an algorithm's performance. In the same spirit as Georgescul et al. (2009), we normalise the T-score and Visit/Post ratio to get two values that we can use a weighted average to combine. In order to do this, we consider again the thread posts and visits as a sequence of events. Any visits that occur after the last post are ignored.

We then consider the worst case in terms of timeliness, or misses. This would be the case where the visit comes at the end, at the same time as the post. So we get a value $T_{\max}$ and P_{miss} such that

$$\begin{aligned} Pr_{\text{miss}} &= \frac{T}{T_{\max}} \\ &= \frac{T}{\left(\frac{\sum_{\rho}(\max_{\rho'} t(\rho') - t(\rho))}{|P|}\right)} \\ &= \frac{|P| \cdot T}{\sum_{\rho}(\max_{\rho'} t(\rho') - t(\rho))} \end{aligned}$$

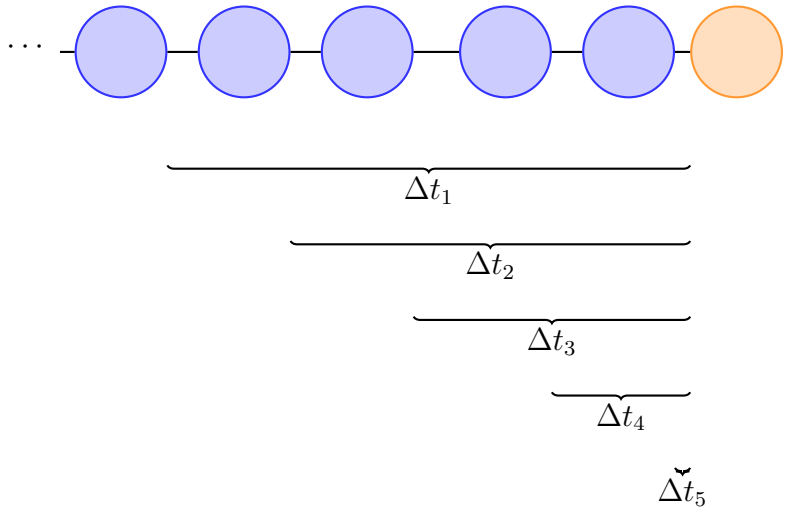


Figure 4.2: An example of calculating $T_{\max}$. A visit is assumed at the same time as the final post made, and the usual T-score metric is calculated

An example can be viewed in Figure 4.2. Assuming that there are no posts before ρ_1 here, we simply take the usual T-score value to get $T_{\max}$ It is difficult to consider the worst case in terms of false alarms, or visits that retrieve nothing. There could be an infinite number of visits made if we are to take the extreme case. In order to get around this, we consider discrete time frames in which a visit can occur. Since for this dataset, our time granularity is in terms of minutes, we shall use minutes as our discrete time frame.

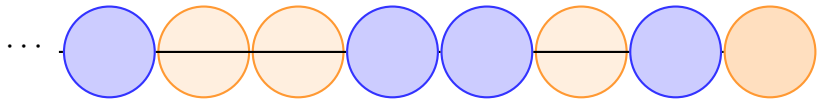


Figure 4.3: An example of calculating the maximum number of visits given a thread.

Using this simplified version of our series of events, we can then imagine a worst-case performing revisit policy that visits at every single time frame. Here, we assume all quantities are measured in terms of minutes. This gives us the following expression

$$Pr_{\text{FA}} = \frac{|V|}{(\max_{\rho} t(\rho)) - |P|}$$

Figure 4.3 shows an example of how Pr_{FA} is calculated.

With these two normalised forms of the original metrics, we can use a weighted mean to give a weighted combined form of the two error rates, Pr_{error}:

$$Pr_{error} = \alpha Pr_{\text{FA}} + (1 - \alpha) Pr_{\text{miss}}$$

In the following chapters, we will discuss the results of our experiments with the various algorithms found in the previous chapter, and measure their effectiveness using the following metrics:

- T**-scores**: the average time taken for a post to be retrieved after it is posted (in minutes)
- **Visit/Post ratio**: the average number of visits required before a post is retrieved.
- Pr_{error}: a combined, normalised score representing the above two metrics.

Chapter 5

Evaluation

In our project, the dataset we used was crawled from `http://www.avsforum.com/f/`. The forum dealt mainly with Audio-Visual equipment, with discussions mainly about technical details, offers and people showing off their DIY projects.

The forum was chosen from the list which Yang et al. (2009) provided in their paper. The forum users use mainly proper English, which made removing stopwords and stemming simpler.

We crawled 4,158 threads, with a total of 1,002,225 posts. A distribution of how the length of threads are distributed can be seen in Figure 5.1. The distribution of the time differences are shown in Figure 5.2. In both the figures, the right-hand-side cutoff was set at 1,000 due to the negligible number of items to the right of the cutoff.

We have also found that the time of the day the day of the week matters when dealing with threads. An example of such a thread can be seen in Figure 5.3a and Figure 5.3b, where we see that activity peaks at 2 PM, dropping slightly during (an assumed) lunch period, and then peaks again during the early evening and at 9 PM. Activity drops to its lowest at 3 AM. The weekly graph also shows a pattern, showing lower posting frequencies during the weekends, and its highest peak on Thursdays.

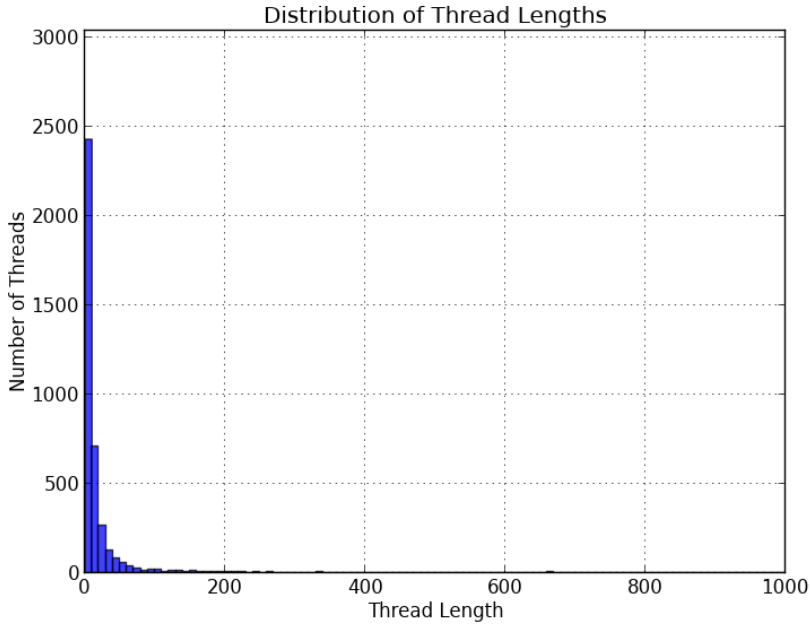


Figure 5.1: Distribution of thread length

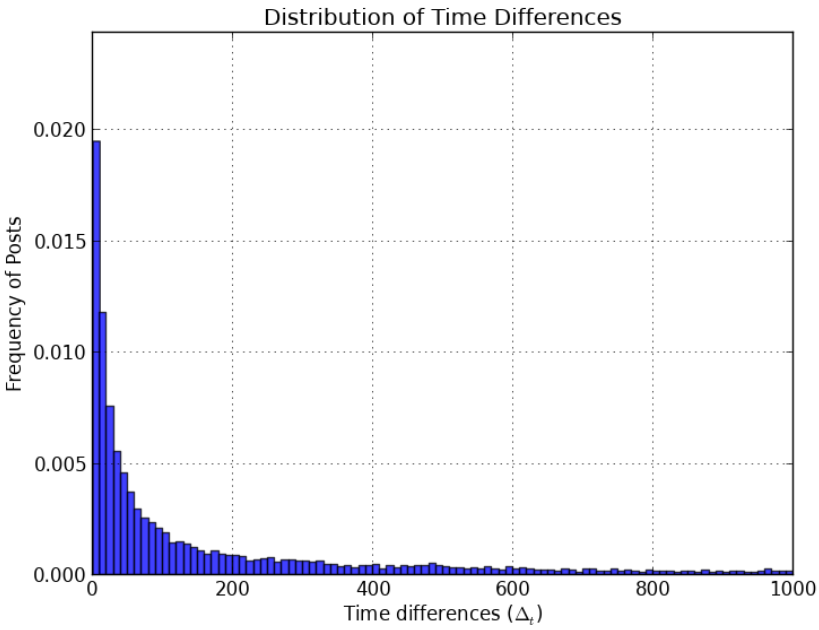
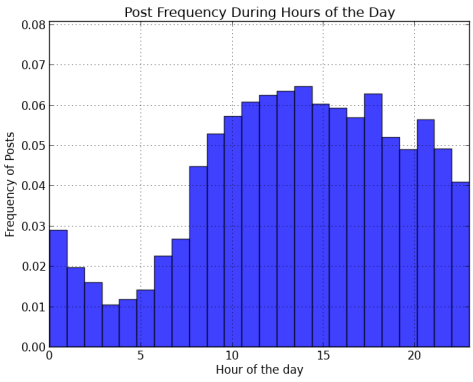
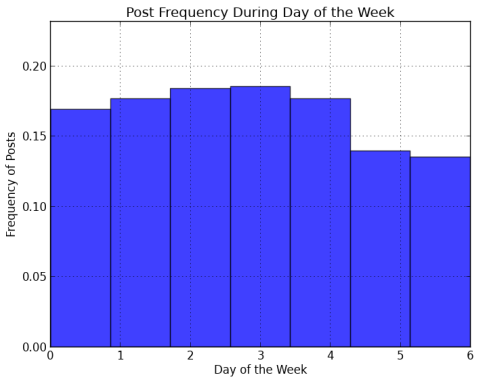


Figure 5.2: Distribution of Δ_t



(a) The hourly post frequency.



(b) The daily post frequency.

5.1 Experiment setup

The first 75% of the thread was used as training data, while the remaining 25% was used as test data. This setup is used in both the tuning of parameters and for the full evaluation of the entire dataset.

5.1.1 Parameter Tuning

Before we begin performing experiments on the full dataset, we first tuned the machine learning algorithms using a sample of the forum threads. In the following experiments, the threads chosen from our extracted dataset are those with a 100 to 1000 posts. This amounted to 97 threads. In each of these experiments, we run the algorithm with different parameters, and use the optimal one in our final evaluation.

Vocabulary size

Before we begin tuning the other parameters, we start with limiting the size of the feature vector when using the content information. This is important as having an excessively large feature vector due to a large vocabulary size leads to complications due to limited memory size. We restrict our search for a good size of vocabulary from $10 \leq K \leq 60$,

	T-score	Visit/Post	Pr_{error}
$K = 10$	1590.985 ± 236.131	18.013 ± 7.588	0.019 ± 0.004
$K = 20$	1594.691 ± 237.001	18.005 ± 7.585	0.019 ± 0.004
$K = 30$	1580.981 ± 235.293	18.011 ± 7.587	0.019 ± 0.004
$K = 40$	1579.544 ± 234.802	18.015 ± 7.586	0.018 ± 0.004
$K = 50$	1564.888 ± 235.164	18.024 ± 7.587	0.018 ± 0.004
$K = 60$	1564.759 ± 234.663	18.017 ± 7.588	0.019 ± 0.004

Table 5.1: Experiment results: Varying vocabulary size

	T-score	Visit/Post	Pr_{error}
$w = 5$	1537.682 ± 234.658	18.056 ± 7.585	0.018 ± 0.004
$w = 10$	1485.157 ± 198.664	18.523 ± 8.028	0.019 ± 0.004
$w = 15$	1433.771 ± 185.080	19.396 ± 8.896	0.016 ± 0.003
$w = 20$	1577.639 ± 229.482	19.037 ± 8.690	0.019 ± 0.004

Table 5.2: Results for tuning parameters using only $\mathbf{t}_\Delta$

with increments of 10. The results are shown in Table 5.1. While $K = 60$ gives us the best T-score, we select $K = 50$ as this gives us the best Pr_{error} score.

Window size

Using a combination of feature sets, we experiment with different window sizes, $w = 1, 5, 10, 15$.

Performing the experiment using only the Δ_t values within the window, we obtain the results found in Table 5.2. The results show that $w = 15$ provide the best T-score. We must however, keep in mind that its Visit/Post ratio is the highest, but also has a higher standard error.

Using only the content, we perform the same experiment again. This gives us the

	T-score	Visit/Post	Pr_{error}
$w = 5$	1593.380 ± 237.070	18.007 ± 7.585	0.019 ± 0.004
$w = 10$	1546.839 ± 198.243	18.493 ± 8.030	0.023 ± 0.006
$w = 15$	1491.695 ± 187.589	19.359 ± 8.899	0.021 ± 0.005
$w = 20$	1645.177 ± 232.365	19.017 ± 8.694	0.024 ± 0.005

Table 5.3: Results for tuning parameters using only $\mathbf{v}$

	T-score	Visit/Post	Pr_{error}
$w = 5$	1537.673 ± 234.657	18.056 ± 7.585	0.018 ± 0.004
$w = 10$	1485.137 ± 198.662	18.523 ± 8.028	0.019 ± 0.004
$w = 15$	1433.762 ± 185.078	19.396 ± 8.896	0.016 ± 0.003
$w = 20$	1577.639 ± 229.482	19.037 ± 8.690	0.019 ± 0.004

Table 5.4: Results for tuning parameters using $\mathbf{t}_\Delta + \mathbf{v} + \mathbf{t}_{\text{ctx}}$

results in Table 5.3. The best T-score here does not do as well as that in the previous experiment. However, it is interesting to note that, again, $w = 15$ results in the best T-score.

For our final experiment for tuning the window size, we combine the various feature sets together. We also include the time-context in this experiment, and we arrive at the results found in Table 5.4. Again, $w = 15$ has the best T-score, but only with a slight improvement over our first experiment.

In any case, this suggests that $w = 15$ may be the best window size. In the following experiments, this will be our w value.

Decay factor

In our discounted sum method, we have to tune the α parameter. We search through 0.1 to 0.9 (inclusive) with increments of 0.1 to find the best possible value for α. We used

	T-score	Visit/Post	Pr_{error}
$\alpha = 1$	1433.761 ± 185.078	19.396 ± 8.896	0.016 ± 0.003
$\alpha = 2$	1433.759 ± 185.078	19.396 ± 8.896	0.016 ± 0.003
$\alpha = 3$	1433.757 ± 185.078	19.396 ± 8.896	0.016 ± 0.003
$\alpha = 4$	1433.755 ± 185.077	19.396 ± 8.896	0.016 ± 0.003
$\alpha = 5$	1433.755 ± 185.077	19.396 ± 8.896	0.016 ± 0.003
$\alpha = 6$	1433.755 ± 185.077	19.396 ± 8.896	0.016 ± 0.003
$\alpha = 7$	1433.755 ± 185.077	19.396 ± 8.896	0.016 ± 0.003
$\alpha = 8$	1433.755 ± 185.077	19.396 ± 8.896	0.016 ± 0.003
$\alpha = 9$	1433.746 ± 185.076	19.396 ± 8.896	0.016 ± 0.003

Table 5.5: Results for tuning α for the `DEC` method

the combined set of features for this experiment. The results are shown in Table 5.5.

$\alpha = 0.9$ performs the best, but its improvement over the rest of the values for α are not by much. Also, note that the T-scores do not defer much from the previous experiment, although there is a slight improvement.

Learning rate for Stochastic Gradient Descent

Because of the scaling factors applied to the sigmoid function, a small change in the exponent of e results in huge fluctuations. As such, we need to find a small enough learning rate such that the predicted values do not end up at only the extremes (λ or Λ), but large enough such that the model is adaptive enough to "react" to changes.

In this experiment, we find that $\eta = 5 \cdot 10^{-8}$ is the best value for the learning rate. Also note that this model produces the best results for the sample dataset.

So at the end of tuning our feature set and parameters, we have the following set of parameters: $K = 50, w = 15, \alpha = 0.9, \eta = 5 \cdot 10^{-8}$. Using these parameters, we run a full evaluation on our dataset.

	T-score	Visit/Post	Pr_{error}
$\eta = 6$	1527.249 ± 196.277	19.231 ± 8.896	0.019 ± 0.004
$\eta = 8$	1379.733 ± 188.469	19.323 ± 8.893	0.015 ± 0.002
$\eta = 10$	1373.679 ± 176.366	18.670 ± 8.966	0.016 ± 0.003
$\eta = 12$	1466.572 ± 226.819	19.267 ± 8.894	0.016 ± 0.003

Table 5.6: Results for tuning η for the `SGD` method

	T-score	Visit/Post	Pr_{error}
`BL`	2370.09 ± 222.556	18.4430 ± 3.009522	0.0455729 ± 0.00315738
`SVR`	1435.96 ± 177.004	21.7333 ± 5.20832	0.0383524 ± 0.00253913
`DEC`	1435.95 ± 177.004	21.7333 ± 5.20832	0.0383523 ± 0.00253913
`SGD`	1881.84 ± 416.264	21.0476 ± 5.05733	0.0380083 ± 0.00247992

Table 5.7: Full evaluation using 830 threads

5.2 Experiments

For our experiments, we selected threads larger than the window size, and have enough posts such that we can split each thread into our 3:1 ratio for training and testing. As such, the threads are at least 19 posts long (5 windows), since the training set also requires at least two posts in order for us to apply our Pr_{error} metric.

This reduces our dataset to a size of 830 threads. We run all three of our algorithms on the dataset, and determine if the performance based on the Pr_{error} is an improvement over the baseline of revisiting at the average rate. The results are seen in Table 5.7.

It is evident that the scores our methods attain are better than the baseline, but only on average. We performed a statistical significance test on our data, using the Wilcoxon signed-rank test (Wilcoxon, 1945). The reason a Student's t-test could not be employed for this test was due to the non-normal distribution of the Pr_{error} results. The Wilcoxon's signed-rank test, like Student's t-test, also gives us a p-value for statistical significance.

	T-score	Visit/Post	Pr_{error}	
SVR	-938.471 ± 161.545	1.29 ± 5.20697	-0.00728408 ± 0.00327361	$p < 0.05$
DEC	-938.474 ± 161.545	1.29 ± 5.20697	-0.00728413 ± 0.00327361	$p < 0.05$
SGD	-479.093 ± 391.269	1.1346 ± 5.19777	-0.0068692 ± 0.00319584	$p < 0.10$

Table 5.8: Paired difference evaluation results

For each of the three proposed methods, we pair the values with the same thread for the baseline, and find the difference in T-score, Visit/Post ratio, and Pr_{error}. We only perform the statistical significance test on Pr_{error}.

We can observe from Table that first two regression methods using SVR perform significantly better than the baseline, while SGD method has a p-value of less than 10%. However, do the models perform differently for different thread lengths?

Different thread lengths

We divide our evaluated threads into 5 different subsets: $0 \leq |P| < 50$, $50 \leq |P| < 100$, $100 \leq |P| < 150$, $150 \leq |P| < 200$ and $|P| \geq 200$. We then look at the same evaluation metrics, and see how well these methods do on different sizes of training data.

The results, displayed in Table 5.9, show that the two methods using an offline trained model perform poorer on longer thread lengths. This suggests that our method that adaptively adjusts its weights based on new observations performs better in the long run, with longer threads. Since SGD only uses content features, this also suggests that the words found in the thread's content can affect the rate of posting on the thread.

The limitations of this, however, are that a sufficient number of observations of posts must be made before the model can make good enough predictions.

	T-score	Visit/Post	Pr_{error}
$\|P\| < 50$			
SVR	1435.96 ± 177.004	21.7333 ± 5.20832	0.0383524 ± 0.00253913
DEC	1435.95 ± 177.004	21.7333 ± 5.20832	0.0383523 ± 0.00253913
SGD	1881.84 ± 416.264	21.0476 ± 5.05733	0.0380083 ± 0.00247992
$50 \leq \|P\| < 100$			
SVR	1554.5 ± 292.206	30.9191 ± 16.996	0.0383057 ± 0.00434776
DEC	1554.5 ± 292.206	30.9191 ± 16.996	0.0383057 ± 0.00434776
SGD	1399.06 ± 254.049	30.6367 ± 16.8814	0.0424276 ± 0.00467794
$100 \leq \|P\| < 150$			
SVR	1700.47 ± 300.413	28.7258 ± 17.6231	0.0148638 ± 0.00365225
DEC	1700.44 ± 300.408	28.7258 ± 17.6231	0.0148638 ± 0.00365225
SGD	1676.24 ± 319.828	28.1005 ± 17.2635	0.0131775 ± 0.00266262
$150 \leq \|P\| < 200$			
SVR	1527.77 ± 349.794	12.4909 ± 6.69594	0.0151237 ± 0.00453188
DEC	1527.77 ± 349.794	12.4909 ± 6.69594	0.0151237 ± 0.00453188
SGD	1223.58 ± 260.261	11.3483 ± 6.03096	0.0143991 ± 0.00369424
$\|P\| \geq 200$			
SVR	708.178 ± 136.94	9.54313 ± 3.14934	0.00642689 ± 0.00124465
DEC	708.175 ± 136.94	9.54316 ± 3.14934	0.00642643 ± 0.00124457
SGD	608.227 ± 109.399	8.45964 ± 2.70039	0.00605408 ± 0.000898222

Table 5.9: Breakdown of evaluation results

5.3 Recommendations

Based on our findings, we can make some suggestions as to how an incremental crawler could use these techniques to predict when to revisit a site.

`SVR` could be used to predict revisitation rates initially, since they work better on shorter threads with $|P| < 50$. When the thread grows sufficiently, `SGD` could then be used.

This has several benefits. Using `SGD`, the weight vector could be loaded into memory when a new observation is made, the weights updated accordingly. In this way, all that needs to be persisted would be the set of weights for every thread.

In this chapter, we have seen that the methods that we propose perform significantly better the baseline, and we have also performed a more fine-grained analysis based on the thread length, and found that the `SGD` method performs slightly better than the other offline learning methods in longer threads.

We must, however, note that the experiment results are based only on one dataset. The results found here may not be general enough to be applied for all forums. Also, `SGD` method is notoriously slow to run, which may be a problem for large feature vectors.

Chapter 6

Conclusion

With the increasing number of sites leveraging user-generated content, a method for predicting the updates of such sites needs to be created in order for an incremental web crawler to effectively crawl the site. Our high level goal: to predict the posting behaviour of users to such sites.

While this primary goal has many challenges, in this report, we have chosen to address challenges specific to forum threads. We want to predict, given content of the current thread, the time at which a user would post to the thread. We also need to ensure that bandwidth consumption is not excessive in the process maintaining the freshness of the crawled data.

We evaluate three different machine learning approaches: Two offline algorithms, one that only takes into account only the latest window, and another that accounts for past windows, with decreasing weightage. And an online algorithm, that uses gradient descent to update its weights every time a new post is observed.

Overall, our evaluation shows that our methods work better than the baseline, which was to revisit the thread at the average time interval. These are promising results, and more can be done to improve upon them.

6.1 Contributions

We have made the following contributions with our work:

1. Provided evaluation metrics that can be parameterised, depending on the evaluators' priority: freshness or bandwidth. The metric allows for an easy way to compare the performance of two models.

2. Proposed three different methods that can be employed for making revisit time estimations. These methods perform significantly better than the baseline. We also made recommendations on how they can be used in a web crawler.

6.2 Future Work

With the proposed methods still having some shortcomings when predicting new posts, or providing insight into what causes post arrival times, it leaves much room for future work to be done.

6.2.1 Topic modelling

One approach we initially considered involved using topic modelling did not pan out due to time constraints (See Appendix A). This approach involved modeling the process of thread posts as a Hidden Markov Model, with topics as the hidden states, producing words and a distribution of time differences. Some work already exists that use HMMs for prediction of data like stock price predictions (González, Muñoz, Roque, & García-gonzález, 2005) and using HMMs together with LDA for making better predictions of language models (Hsu & Glass, 2006).

6.2.2 Using Natural Language Processing (NLP) techniques

An improvement could be made to the feature set used in our project. A source for features that could be useful may be found in Wang et al. (2012). Since their work aims to predict retweetability of tweets, the same features being considered could also play a role in determining when a new post is made.

6.2.3 Leveraging context

One of the assumptions made in selecting the feature set as we did, was that the thread was self-contained: the content of the thread will affect the rate of posting. However, we know this is not true, since news regarding a certain product may affect a discussion thread of said product.

If some data from recent news could be factored into the features, this may lead to a better prediction of post arrival times.

6.2.4 Using adaptive learning techniques

An interesting approach to forecasting stock prices was presented in Cao and Tay (2003). The technique involved tweaking conventional SVMs to weigh recent training instances more heavily than older instances. This is a particularly useful idea, since we face the same issue in our task: Recent posts are more descriptive of the current state of the thread, and hence should be more useful in predicting the next post.

References

Brewington, B. E., & Cybenko, G. (2000). Keeping up with the changing web. *Computer*, , 2000, 52–58.

Cao, L., & Tay, F. (2003). Support vector machine with adaptive parameters in financial time series forecasting. *Neural Networks, IEEE Transactions on*, *14*(6), 2003, 1506–1518.

Cho, J. (1999). The evolution of the web and implications for an incremental crawler. *Science*, , 1999, 1–18.

Cho, J., & Garcia-Molina, H. (2003). Effective page refresh policies for Web crawlers. *ACM Transactions on Database Systems*, *28*(4), December, 2003, 390–426.

Cho, J., & Garcia-molina, H. (2003). Estimating Frequency of Change. (650), 2003.

Coffman, E., & Liu, Z. (1997). Optimal robot scheduling for web search engines. *Sophia*, , 1997.

Drucker, H., Burges, C., Kaufman, L., Smola, A., & Vapnik, V. (1997). Support vector regression machines. *Advances in neural information processing systems*, , 1997, 155–161.

Georgescul, M., Clark, A., & Armstrong, S. (2009). An analysis of quantitative aspects in the evaluation of thematic segmentation algorithms. *... of the 7th SIGdial Workshop on ...*, (July), 2009, 144–151.

González, A. M., Muñoz, A., Roque, S., & García-gonzález, J. (2005). Modeling and Forecasting Electricity Prices with Input / Output Hidden Markov Models. *20*(1), 2005, 13–24.

Hsu, B.-j. P., & Glass, J. (2006). Style & Topic Language Model Adaptation Using HMM-LDA. (July), 2006, 373–381.

Naaman, M., Boase, J., & Lai, C.-h. (2010). Is it really about me?: message content in social awareness streams. *Proceedings of the 2010 ACM conference ...*, , 2010, 189–192.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, *12*, 2011, 2825–2830.

Sriram, B., & Fuhry, D. (2010). Short text classification in twitter to improve information filtering. *... in information retrieval*, , 2010, 4–5.

Tan, Q., Zhuang, Z., & Mitra, P. (2007). Designing efficient sampling techniques to detect webpage updates. *Proceedings of the 16th*, *1*(3), 2007.

Wang, A., Chen, T., & Kan, M. (2012). Re-tweeting from a Linguistic Perspective. *NAACL-HLT 2012*, , 2012.

Wang, L., & McCarthy, D. (2011). Predicting Thread Linking Structure by Lexical Chaining. *Proceedings of the*, , 2011, 76–85.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, *1*(6), 1945, 80–83.

Yang, J., Cai, R., Wang, C., & Huang, H. (2009). Incorporating site-level knowledge for incremental crawling of web forums: A list-wise strategy. *on Knowledge*, , 2009, 1375–1383.

Appendix A

Topic Modelling

A.1 Introduction

In this project, we plan to use topic modelling to predict the arrival times of new posts.

Using Latent Dirichlet Allocation, we want to find a set of topics with different time interval distributions. These topics also have a probability from transitioning to other topics as the thread continues. What we want to do is, based on the current post content, to be able to get a distribution of which state/topic the thread is currently in, and using the time interval distribution, predict the arrival time of the next post

We shall call the time differences between posts Δ_t. For the rest of this report, $\mathbf{W}$ represents a document, in particular, the collection of words that the document contains, and $\mathbf{Z}$ represents a topic that a document belongs to.

In the next section, we discuss the details of the project that have been completed and some of the complications. The last section will detail the remaining tasks left to be done.

A.2 Completed Tasks

A.2.1 Data: avsforum.com

Our dataset was obtained from from `avsforum.com` and stored in a tab-delimited format, with each line representing a new post. For each post, we extracted the following:

Timestamp Time in seconds-from-epoch format.

Author Username of the author of the post.

Main content The main textual content of the post.

For the our purposes, we pre-process the data such that each instance is comprised of k posts. We have found that Support Vector Regression gives the best results when trying to infer the time of the next post arrival when $k = 15$.

A.2.2 Latent Dirichlet Allocation

We used Gibbs sampling to perform LDA for 2 to 10 topics. For a small subset of threads in the forum. Treating each window as a document, we attempted to find topics that the words in the documents belonged to. The following are some interesting results.

Since it is a forum that deals largely with audio visual equipment, a large portion of the posts tend to comprise of people seeking to troubleshoot faulty equipment. One of the topics LDA found had the following words:

```
player dvd get one would use play audio like samsung blu hdmi
sound work problem ray also tri think firmwar set disc know issu
new see updat look want soni good better time connect even
thankstill need say buy receiv back price output oppo make seem
video unit realli support well soundbar differcould vizio optic
format movi analog come cabl via got anyon much sinc denon
```

```
may sure display decod thing doesnbest way bar fix pcm post first
watch channel hope test digit said far take panason right
anoth bitstream return abl file
```

These seem to be the keywords that are consistent with such posts – when users describe 'issues' and 'problems' with their 'firmware', and decide if they should 'return' the goods.

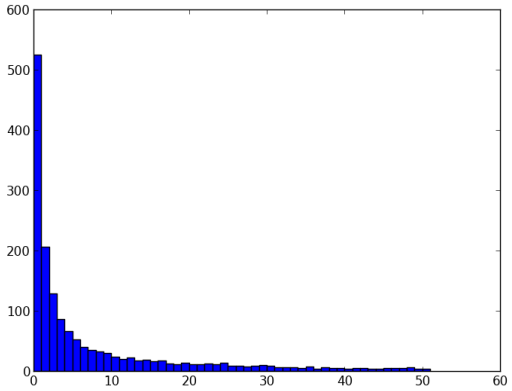
Another distinct topic that was revealed had to do with positive sentiment, and contain words such as 'thanks' or 'nice'.

```
speaker like amp sub sound one #### use ### would get listen room
music good also look power two system realli set think much
know post channel better new pair audio great thank bass time
hear heard differ even make see high want say well center
come level surround back could still price current wire need
home jamo dsp first play tri never sure nice love end
output peopl design receiv thread focu run cabl front review
theater test #.# may anyon year thx got hsu watt legaci compar
subwoof setup main bit work right klipsch somethdual impress guy
```

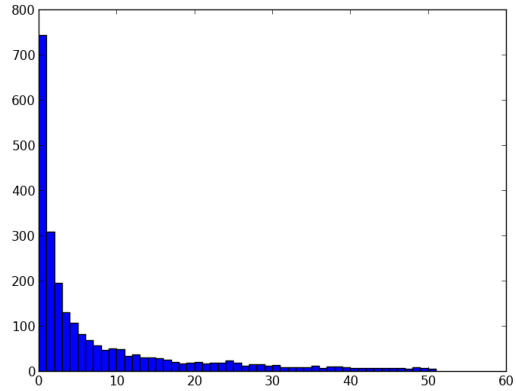
However, what we want to study is if these differences in content reflect a difference in the arrival times of the next post.

A.2.3 Distribution of Δ_t

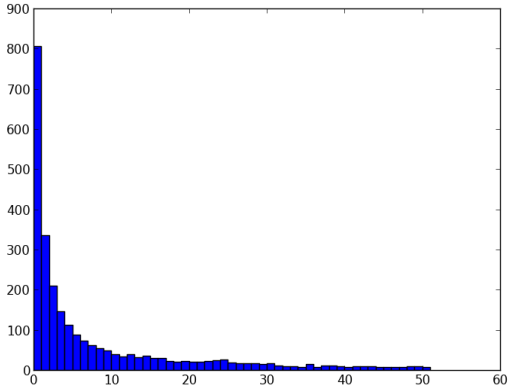
Since we have a way of obtaining $Pr\mathbf{ZW}$, we need $Pr\Delta_t\mathbf{Z}$ in order to have a way to predict Δ_t. To have a sense of what this looks like, we have binned the Δ_t into bins of 20 minutes, and plotted their frequency based on the topics we extracted. A document is said to be coming from topic z if $\max_{z'} Prz'\mathbf{W} = z$ The idea was to see if there was a distinguishable Δ_t distribution given different topics. See Figure A.1.



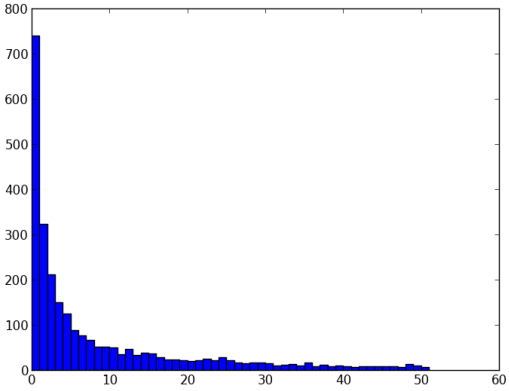
(a) Topic 1



(b) Topic 2



(c) Topic 3



(d) Topic 4

Figure A.1: Time distributions for $k = 4$

Unfortunately, the time distributions did not appear very distinct from each other. More tests need to be performed to see if using the expected value from this currently gathered data will reflect anything.

A.3 Work In Progress

Including the previous post topic, we can make a better estimate of the Δ_t distribution of the current post.

$$p(\mathbf{Z}_t|\mathbf{Z}_{t-1}, \mathbf{W}) = \frac{p(\mathbf{Z}_{t-1}|\mathbf{Z}_t) \cdot p(\mathbf{Z}_t|\mathbf{W}_t) \cdot p(\mathbf{W}_t)}{p(\mathbf{Z}_{t-1}) \cdot p(\mathbf{W})}$$

This essentially makes it similar to a Hidden Markov Chain, since the document topic is dependent on the previous document's topic, and the observations produced are the text. This is, however, dependent on whether the distribution of Δ_t is different for each topic.

As seen in Figure A.1, which shows one example of LDA produced topics, the time distributions are not distinguishable. This may suggest that this approach may not be feasible, and more work needs to be done.