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

In this report, we study a problem and design an efficient algorithm to solve the problem. We implemented the algorithm and evaluated its performance againts previous proposed algorithms that solves the same problem. Our results show that our algorithm runs faster.

Subject Descriptors:
C5 Computer System Implementation
G2.2 Graph Algorithms

Keywords:
Problem, algorithm, implementation

Implementation Software and Hardware:
python, bash

Acknowledgement

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. In this project,our goal is to predict updates in such threads.

Table 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. While mining structured, curated content from sites like Amazon, for data like prices is easy and effective, data that can be obtained from user-generated content are of a different nature. One may be able to infer public sentiment about a given product that would not be readily available from an e-commerce site. In some cases, news may travel more quickly through such online community discussion than through traditional media. Users also typically discuss purchased products bought online via these forums, and companies that want to get timely feedback about their product should turn to data mined from such sites.

A naive way of getting timely updates is to aggressively hit the pages repeatedly downloading the pages at a very frequent rate. However, the number of pages in a forum site are far too large to perform this efficiently on every forum. One way to minimise this cost would be to look at the time differences between previous posts to estimate the arrival of the next one. 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.

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

	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

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. Suppose 10 days later, someone with a slight variation of the same problem posts on the thread again. A crawler that estimates update rates solely on the age of the thread to determine its download rate of the thread may not update itself with the thread.

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 (1) incur excessive costs when downloading un-updated pages, and (2) raise the possibility of the web master blocking the requester's IP address.

This year-long project proposes to use content-based features of a given thread to predict its next update time. We argue, that the content within the posts of the thread should be important in predicting the thread updates, and propose our approach to solving the problem.

Chapter 2

Related Work

2.1 Refresh policies for incremental crawlers

In order to devise such a strategy, we need to predict how often any user may update a 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 will 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 and threads for postprocessing later. 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 et. al. 2009 (Yang, Cai, Wang, & 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. Coffman et. al. (Coffman & Liu, 1997) analysed the theoretical aspects of doing this, showing that if the page change process is governed by a Poisson process $\lambda e^{-\lambda\mu}$, 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 4 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 et. al. (Tan, Zhuang, & Mitra, 2007) and Wolf et. al. (Wolf, Squillante, & Yu, 2002). However, the Poisson distribution is memoryless, and in experimental results due to Brewington and Cybenko (Brewington & 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 may be very different from that of static pages.

Yang et. al. (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 predict when the next update to the thread will arrive.

Online forums and bulletins have a logical, hierarchical structure in their layout, which typically alerts the user to thread updates by putting threads with new replies at the very top of the thread index. Yang's work exploits this as well as their linear model to achieve a predicton of when to retrieve the pages. However, this is not so for comments found on blog sites or discussion threads in an e-commerce site about a certain product and the lack of these pieces of information may result in a poorer estimate, or no estimate at all.

Our perspective is that the available content on the thread at the time of the retrieval should also be factored into the model used to predict the 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 will review some existing work related to analysing thread-based pages which we think will aid us in our efforts to do content-based prediction.

Wang et. al (Wang & McCarthy, 2011) did work in finding out linkages between forum posts using lexical chaining. They proposed a method to link posts using the tokens in the posts called $Chainer_{SV}$. While they do analyse the content of the individual posts, the paper does not make any prediction with regards to newer posts. The methods used to produce a numeric similarities between posts may be used as a feature to describe a thread in its current state, but incorporating this into our model is non-trivial.

With these related work in mind, we next propose our modelling of a thread as a Markov chain, and our approach to solving the problem.

Chapter 3

Method

Our project aims to predict the amount of time between the arrival of the next post and the time the last post in the thread was made. The information available to us are the previously made posts that we observe when first visiting the thread. 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.

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 $\rho_{t-w+1}, \rho_{t-w+2}, \cdots, \rho_t$, we can approximate Δ_t with $f(\mathbf{X})$. In the following sections, we will discuss various methods for estimating f. Various notations will be used, a quick reference is provided in Table 3.1.

We extracted the timestamp, author and text content for each post in each thread from the forums. Our dataset was obtained from from `avsforums.com`.

Previous time differences All the time differences between posts made in the window. ($\mathbf{t}_\Delta$)

Time-based features Day of week, Hour of day. Provides contextual information about when the post was made. ($\mathbf{t}_{\text{ctx}}$)

Content features (text) Word frequency counts are used for this set of experiments. Using regression, we find the top K variables that the actual Δ_t depends on. Table 3.2 reflect the results of the experiments done with varying values of K.

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

Table 3.1: Notation reference

	MAPE	Pr_{miss}	Pr_{fa}	Pr_{error}	T-score	Posts	Visits
$w = 5, \mathbf{v}, \lvert\mathbf{v}\rvert = 5$	8.441	0.922	0.064	0.493	1601.321	33.000	545.371
$w = 5, \mathbf{v}, \lvert\mathbf{v}\rvert = 10$	8.632	0.924	0.064	0.494	1593.763	33.000	545.206
$w = 5, \mathbf{v}, \lvert\mathbf{v}\rvert = 15$	8.913	0.924	0.064	0.494	1594.276	33.000	545.381
$w = 5, \mathbf{v}, \lvert\mathbf{v}\rvert = 20$	9.382	0.923	0.063	0.493	1597.533	33.000	545.495
$w = 5, \mathbf{v}, \lvert\mathbf{v}\rvert = 25$	9.905	0.927	0.063	0.495	1597.295	33.000	545.619
$w = 5, \mathbf{v}, \lvert\mathbf{v}\rvert = 30$	10.836	0.925	0.063	0.494	1587.734	33.000	545.619
$w = 5, \mathbf{v}, \lvert\mathbf{v}\rvert = 35$	12.044	0.927	0.064	0.495	1608.698	33.000	545.722
$w = 5, \mathbf{v}, \lvert\mathbf{v}\rvert = 40$	12.400	0.927	0.063	0.495	1593.062	33.000	545.649
$w = 5, \mathbf{v}, \lvert\mathbf{v}\rvert = 45$	12.462	0.928	0.063	0.496	1587.913	33.000	545.649
$w = 5, \mathbf{v}, \lvert\mathbf{v}\rvert = 50$	12.994	0.926	0.063	0.495	1581.241	33.000	545.804

Table 3.2: Experiment results: Varying vocabulary size

3.1 Baselines

A simple way of estimating the revisit rate would be to use the average time differences given the observed posts, or a training set. In previous work, we have seen that if page updates follow a Poisson distribution, then revisiting at the Poisson mean would be an optimal revisit policy.

There are some details that need to be considered. One revisit policy would be to do so at a constant, fixed rate, independent of the posts being made to the thread. For our baseline revisit policy, we took into account the last made post whenever we revisit, and calculate our next revisit time based on the last post.

Another way of predicting using average post interval timings would be to use a sliding window. Averaging out the time differences between the posts would intuitively work, because it captures the context of the situation: A series of posts with short intervals should mean that the next post would come at around the same interval as the few that came before.

3.2 Performing regression on windows

Previous work has used linear regression on a number of different features extracted from forums (Yang et al., 2009). In their paper, the regressed function was used as a scoring function rather than a predictive function. In site of this, we attempted to implement the same model, but this resulted in evaluations worse than that of the baseline.

We did use some of the features mentioned in the paper: 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 in stead of linear regression, we used a regression method known as Support Vector Regression (SVR). This method allows the using of different kernels, allowing for better estimation of the target function.

The main focus of study in this report was to see if content helps with predicting thread updates would produce an improvement. Some of the ways that content data were extracted into feature vectors are the following: Word frequency, the tf-idf of these words, and Part-of-Speech tags.

We perform the standard preprocessing steps like removing stopwords and tokens of length less than three. We also use Porter's stemming algorithm as another preprocessing step, before performing a word frequency count.

These feature sets are used in different combinations, with different window sizes. The results will be seen in the next chapter.

3.3 Discounted sum of previous instances

The current method uses only information on the current w posts. However, 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.

Following this intuition we attempt to use 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, and $\mathbf{v}$ is the word frequency vector. α is the *discount factor* and satisfies $0 \leq \alpha < 1$.

3.4 Stochastic Gradient Descent

We attempt to use stochastic gradient descent to estimate the function f. However, during runtime, instead of using a static function, we continue to allow f to vary whenever new posts and their update times are observed. Since $f(\mathbf{X}_{t-w}, \ldots, \mathbf{X}_{t-1}) > 0$, 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, becoming exceedingly huge. 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.

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.

Chapter 4

Evaluation

4.1 Potential errors

To be thorough, let us also enumerate the types of errors that a model making predictions could encounter.

The model can potentially make a prediction such that the next visit comes before the arrival of the next post. The predictions being made are the Δ_t between the posts, rather than the visitation times, hence, it is possible for the model to make a prediciton that occurs before the current time. An erroneous prediction can also cause the crawler to come in before the next post (two, or more, visits, but nothing new fetched). Errors of this type waste bandwidth, since the crawler will make an unneccessary visit to the page.

Another type of error would have the prediction causing the next visit to come some time after a post. Since most predictions are almost never fully accurate, there will be some time between the post is made and hte page is fetched. These errors are still relatively acceptable, but the time difference between the post arriving and the visit should be minimised. The visit could also come more than one post later. Errors of this kind incur a penalty on the freshness of the data, more so than the after one post, especially if the multiple posts are far apart time-wise.

In the following experiments, the threads chosen from our extracted dataset are those with a 100 to 1000 posts. This amounted to 97 threads. The first 75% of the thread was u sed as training data, while the remaining 25% was used as test data. We used Support Vector machines

for this regression task, employing a Radial Basis Function kernel as our learning algorithm.

The SVR module from the Python library scikit-learn was used in the implementation of this experiment.

4.2 Evaluation metrics

We use *Mean Absolute Percentage Error* (MAPE), to measure the performance of the learnt model. This value is given by

$$\frac{1}{N}\sum_{i=1}^{N}\left|\frac{A_i - F_i}{A_i}\right|$$

where A_i is the actual value, and F_i is the forecasted value for the instance i. Realistically, the model would not be able to come into contact with every possible window, since chances are it will make an error that causes it to visit a thread late, causing it to miss two posts or more. This value does not reflect how well the model will do in a real-time setting, but gives an idea of how far off the model is given a window.

We also want to know the *timeliness* of the model's visits. Yang et. al. (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

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

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.

$$Pr_{miss} = \frac{\sum_{i=1}^{N-k} [\Theta_{ref\_hyp}(i,k)]}{\sum_{i=1}^{N-k} [\Delta_{ref}(i,k)]} \qquad \Delta_{ref}(i,k) = \begin{cases} 1, & \text{if } r(i,k) > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\Theta_{ref\_hyp}(i,k) = \begin{cases} 1, & \text{if ends with post} \\ 0, & \text{otherwise} \end{cases}$$

$$Pr_{fa} = \frac{\sum_{i=1}^{N-k} [\Psi_{ref\_hyp}(i,k)]}{N-k} \qquad \Psi_{ref\_hyp}(i,k) = \begin{cases} 1, & \text{if ends with visit} \\ 0, & \text{otherwise} \end{cases}$$

Split the Pr_{error} measure into 3 parts:

Probability of having visits before the first post in the window. Probability of having more visits than posts after the first post. Probability of having more posts than visits after the first post.

Viewing the posts made during the thread's lifetime as segmentations of the thread, and the visits made as hypotheses of where the segmentations are, we use the Pr_{error} metric from Georgescul et. al., 2006 as a measure of how close the predictions are to the actual posts. An example can be seen in Figure 4.1.

Function that gives me:

Increase in visit to post ratio increase Increase in interval increase

Increase between post and visit increase (0,∞ or 1)

Increase between visit and visit decrease (∞ or 1,0) Increase between post and post increase (0,∞ or 1)

$$T = \frac{\sum_{e=1}^{|E|-1} \Psi(e_t, e_{t+1})}{|E|-1} \qquad \Psi(e_t, e_{t+1}) = \begin{cases} 1 - e^{-(e_{t+1}-e_t)} & \text{if post,visit} \\ e^{-(e_{t+1}-e_t)} & \text{if visit,visit} \end{cases}$$

4.3 Normalising the T-score and Visit/Post ratio

We normalise the T-score to get a comparable metric across all the threads. In order to do this, we consider again the thread posts and visits as a sequence of events. We then define the *lifetime*, denoted as l, of the thread as the time between the first post and the last post. Any visits that occur after the last post are ignored.

$$\uparrow \quad p_1, \underbrace{\quad p_2, \quad p_3, \quad p_4, \uparrow}_{\text{sliding window}} \uparrow$$

Figure 4.1: An example of the sliding window metric. The metric is made up of two components: First, the probability that, given at least one post is present in the window, there are more visits than post. Secondly, the probability that there are more posts than visits for a given window. The weighted sum of this gives the overall Pr_{error}

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

$$P_{\text{miss}} = \frac{T}{T_{\max}} = \frac{N \cdot T}{\sum_p l - p_t}$$

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.

With 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. This gives us

$$P_{\text{FA}} = \frac{|V|}{l - |P|}$$

where l is in units of our specified discrete time frame.

Chapter 5

Conclusion

5.1 Future Work

References

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

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.

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

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

Wolf, J., Squillante, M., & Yu, P. (2002). Optimal crawling strategies for web search engines. *on World Wide Web*, , 2002.

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

Code