Exploiting Multiple Features with MEMMs for Focused Web Crawling

Hongyu Liu, Evangelos Milios, and Larry Korba

National Research Council Institute for Information Technology, Canada
Faculty of Computer Science, Dalhousie University, Canada
hongyu.liu@nrc-cnrc.gc.ca,eem@cs.dal.ca,larry.korba@nrc-cnrc.gc.ca

Abstract. Focused web crawling traverses the Web to collect documents on a specific topic. This is not an easy task, since focused crawlers need to identify the next most promising link to follow based on the topic and the content and links of previously crawled pages. In this paper, we present a framework based on Maximum Entropy Markov Models(MEMMs) for an enhanced focused web crawler to take advantage of richer representations of multiple features extracted from Web pages, such as anchor text and the keywords embedded in the link URL, to represent useful context. The key idea of our approach is to treat the focused web crawling problem as a sequential task and use a combination of content analysis and link structure to capture sequential patterns leading to targets. The experimental results showed that focused crawling using MEMMs is a very competitive crawler in general over Best-First crawling on Web Data in terms of two metrics: Precision and Maximum Average Similarity.

Key words: Focused Crawling, Web Search, Feature Selection, MEMMs

1 Introduction

General search engines are not always sufficient to satisfy all needs. To address specialized search needs, general search engines and crawlers are being evolved, leading to personalization of search engines, for example, *MyAsk, Google Personalized Search, My Yahoo Search*; localization of search engines, for example, *Google Local, Yahoo Local, Citysearch*; topic-specific search engines and portals, for example, *Kosmix, IMDB, Scirus, Citeseer*. The success of topic-specific search tools depends on the ability to locate topic-specific pages on the Web while using limited storage and network resources. This can be achieved if the Web is explored by means of a *focused crawler*. A focused crawler is a crawler that is designed to traverse a subset of the Web for gathering only documents on a specific topic, instead of searching the whole Web exhaustively. The challenge in designing a focused crawler is to predict which links lead to target pages. Focused crawler can only use information gleaned from previously crawled pages to estimate the relevance of a newly seen URL, therefore, the effectiveness of the focused crawler depends on the accuracy of this estimation process.

A variety of methods for focused crawling have been developed and focused crawling algorithms can be roughly categorized along two different dimensions:

local-feature based and path based. The underlying paradigm of local-feature algorithms is to train a learner with only *local* features collected about relevant nodes *alone*(i.e., the parent pages and sibling pages). These works include Fish-Search, Shark-Search, URL Ordering[1], focused crawler[2], intelligent crawling[3], InfoSpiders[4], generic programming[5], ontology approach[6], and classification[7–9]. Path based algorithms include reinforcement learning[10] and Context Graph algorithm[11]. Both methods capture *longer* path information leading to targets rather than relevant nodes alone, as was the case with the local-feature based crawler. In [10], crawlers are modeled as autonomous agents to learn to choose optimal actions to achieve their goal. The Context Graph method[11] uses the text of page u to estimate the link distance from u to some target pages. Documents classified into layers closer to the target are crawled first. However, two issues remain to be addressed. One is that the assumption that all pages in a certain layer centered at a target document belong to the same topic described by a set of terms does not always hold. Second, there is no discrimination among different links on a page. Since only a fraction of outlinks from a page are worth following, offering additional guidance to the crawler based on local features in the page to rule out some unimportant links can be helpful.

Our approach is to model focused crawling as a sequential task, over an underlying chain of hidden states, defined by hop distance from targets, from which the actual documents are observed. In this paper, we extend our work[12] to exploit multiple overlapping features, such as title, anchor text, and URL token, with Maximum Entropy Markov Models(MEMMs) to represent useful context including not only text content, but also linkage relations.

2 MEMMs

Maximum Entropy Markov Models(MEMMs)[13] are probabilistic sequence models that define conditional probabilities of state sequences given observation sequences. Formally, let o and s be random variables ranging over observation sequences and their corresponding state (label) sequences respectively. We use $s = s_1, s_2, ..., s_n$ and $o = o_1, o_2, ..., o_n$ for the hidden state sequence and observation sequence respectively, where s and o have the same length n. State s_t depends on observations o_t and previous state s_{t-1}. MEMMs are discriminative models that define the conditional probability of a hidden state sequence s given an observation sequence o, $p(s|o)$.

Let n be the length of the input sequence, m be the number of features. MEMMs make a first-order Markov independence assumption among states, that is, the current state depends only on the previous state and not on any earlier states, so $p(s|o)$ can be written as:

$$p(s|o) = \prod_{t=1}^{n} p(s_t|s_{t-1}, o_t) \tag{1}$$

where t ranges over input positions $1..n$. Applying the maximum entropy principle, we can rewrite $p(s_t|s_{t-1}, o_t)$ as the following:

$$p(s_t|s_{t-1}, o_t) = \frac{1}{z(o_t)} \exp(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t)) \quad (2)$$

$$z(o_t) = \sum_{s' \in S} \exp \sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s', o_t) \quad (3)$$

where, s_t is the state at position t, o_t is the observation at position t, the f_i are arbitrary features, λ_i is the weight of the feature f_i, and S indicates a set of all possible states. $z(o_t)$ is called the per-state normalizing factor.

The use of feature functions allows arbitrary, non-independent features in the observation sequence o. The weights λ are the parameters of the model. Training an MEMM involves maximizing the conditional probability, $p(s|o)$, to find the best set of feature weights $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$.

3 Proposed Approach

We model focused crawling as a sequential task and learn the sequential linkage patterns along paths leading to relevant pages by using a combination of content analysis and link structure of such paths. To capture such sequential patterns, we propose to apply MEMM, where the hidden states are based on hop distance from the target and observations consist of the values of a set of pre-defined feature values of observed pages, such as anchor text, URLs, and keywords extracted from the pages.

3.1 Structure of MEMMs for Focused Crawling

Let k be the number of hidden states. The key quantities associated with MEMM are the hidden states, observations(features), and the parameters(λ). Fig. 1 shows the structure of MEMM for focused crawling.

- Hidden states: $S = \{T_{k-1}, T_{k-2}, ..., T_1, T_0\}$. The focused crawler is assumed to be in state T_i if the current page is i hops away from a target. The state T_{k-1} represents "$k-1$" or more hops to a target page.
- Observations: Collections of feature values of page sequences $O = \{page_1, page_2, page_3, ...\}$. Observable page sequences represented by a sequence of values for a set of predefined feature functions $f = \{f_1, f_2, ..., f_m\}$. m is the number of feature functions.
- Set of parameters $\lambda = \{\lambda_1, \lambda_2, ... , \lambda_m\}$, where λ_i is associated with feature function f_i.

3.2 Training MEMMs

MEMMs have parameters $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$ which are the weights for each feature function $f_1, f_2, ..., f_m$. Training means estimating these parameters from

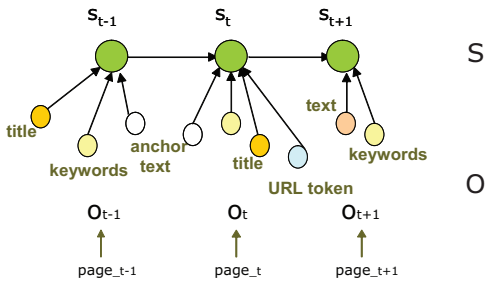


Fig. 1. Dependency structure of MEMMs on modeling a sequence of Web pages. Directed graphical model, arrow shows dependency (cause).

the training data. Given the training data D consisting of N state-observation sequences, $D = \{\mathbf{S}, \mathbf{O}\} = \{(s^j, o^j)\}_{j=1}^N$ (we use superscript j to represent training instances), where each $o^j = \{o_1^j, o_2^j, ..., o_n^j\}$ is a sequence of observations with length n, and each $s^j = \{s_1^j, s_2^j, ..., s_n^j\}$ is the corresponding sequence of states. The task of training MEMMs is to choose values of parameters $\{\lambda_i\}$ which maximize the log-likelihood, $L'_\lambda = \log p(\mathbf{S}|\mathbf{O})$, of the training data. We use (s^j, o^j) to represent the j^{th} state-observation sequence from the training data set, s_t^j, o_t^j to indicate the state and observation at position t of the j^{th} state-observation sequence respectively. The objective function can be written as:

$$L'_\lambda = \log p(\mathbf{S}|\mathbf{O}) = \sum_{j=1}^{N} \log p(s^j|o^j)$$
$$= \sum_{j=1}^{N} \left(\sum_{t=1}^{n} \sum_{i=1}^{m} \lambda_i f_i(s_{t-1}^j, s_t^j, o_t^j) \right) - \sum_{j=1}^{N} \log \prod_{t=1}^{n} \sum_{s' \in S} \exp \sum_i \lambda_i f_i(s_{t-1}^j, s', o_t^j)$$

To perform the optimization of L'_λ with respect to λ, we consider the gradient of the log-likelihood L'_λ and set it to zero. Parameter estimation in MEMMs uses the Limited Memory Quasi-Newton Method (L-BFGS)[14, 15] to iteratively estimate the model parameters λ.

3.3 Training Data Collection

Collecting training data consists of two extraction processes: Local web graph extraction and Page sequence extraction.

Selection of Target Pages on Topics We collect three kinds of data about the topics: keywords, descriptions and target pages. Keywords are formed by concatenating the words appearing in the different levels along the topical hierarchy directory from the top. For example, "Home, Gardening, Plants, House Plants" are extracted keywords for the topic. We also further extract important words embedded in the target pages themselves, including words from title

and headers(<title>...</title>, <h1>...</h1> etc.) and keywords and descriptions from meta data(<meta>...</meta>). Descriptions are generated using the descriptive text and the anchor text in the page of the topic in the Open Directory Project(ODP)[1]. We select topics from an existing hierarchical concept index such as the ODP, and pick topics which are neither too general nor too specific. Our criteria for selecting topics is to pick the topics under 4 or 5 level of ODP topic hierarchy. An example of such a topic is *House Plants*, to be found under the ODP topic hierarchy *Home - Gardening - Plants - House Plants*.

Extraction of the Local Web Graph In order to capture the page sequences leading to targets for training, first we construct a local Web graph to represent the content and linkage structure associated with the known targets. The process of Local Web graph extraction takes specified target pages as input, and builds the local Web graph by following the inlinks to target pages using an inlink retrieval service (as offered by the Yahoo! or Google search engines).

To construct a local Web graph, each Web page is represented by a node, and all hyperlinks between pages are added as edges between the nodes. When a new Web page is found and added to the existing graph, a new node will be created and all the hyperlinks between it and existing nodes will be added into the Web graph as edges. The local Web graph is created layer-by-layer starting from one or more user-specified target pages. Note that we only use ODP to select initial target pages on the topic, rather than using the ODP hierarchy for training. We use the inlink service from Yahoo Web API[2], which retrieves Web pages that have links to the specified Web page. Starting from layer 0 corresponding to user-specified target page(s), the graph is created layer by layer, up to layer 4.

Extraction of Page Sequences and State Sequences Page sequences are extracted directly from the constructed local Web graph. We extract sequences in a random manner: the process starts with a randomly-picked node in layer 4, randomly selects one of its children for the next node, and repeats until a randomly-generated sequence length between 2 and 10 is reached. The following rules are considered for the page sequence extraction:

- Only extract pages from higher layers to lower layers or from the same layer, without reversing the layer order in the sequences.
- Avoiding loops in the sequence.

The complete training data includes page sequences and their corresponding layer sequences. Page sequences are referred to as observation sequences or observable input sequences, and layer sequences are referred to as hidden state sequences or state sequences. The hidden state for a page in our system is its lowest layer number.

[1] http://www.dmoz.org
[2] http://developer.yahoo.com/search/siteexplorer/V1/inlinkData.html

3.4 Focused Crawling

After the learning phase, the system is ready to start focused crawling on the real Web to find relevant pages based on learned parameters. The crawler utilizes a queue, which is initialized with the starting URL of the crawl, and keeps all candidate URLs ordered by their visit priority value. The crawling respects the Robot Exclusion Protocol and distributes the load over remote Web servers. The crawler downloads the page pointed to by the URL at the head of the queue, extracts all the outlinks and performs feature extraction. The predicted state for each child URL is calculated based on the current observable features and corresponding weight parameters, and the visit priority values are computed accordingly. The start page are picked randomly from the 4^{th} level of Web graph created using backlink service for training, arranging from 2 to 6.

Efficient Inference We now discuss two kinds of inference we are going to use in Focused Crawling stage. When the crawler sees a new page, the task of the inference is to estimate the probability that the page is in a given state s based on the values of all observed pages already visited before. We are using two major approaches to compute the probabilities in our experiments: marginal probability and the Viterbi algorithm, and they can be performed efficiently using dynamic programming.

`Marginal Probability` – The marginal probability of states at each position t in the sequence is defined as the probability of states given the observation sequence up to position t. Specifically, the forward probability, $\alpha(s,t)$ is defined as the probability of being in state s at position t given the observation sequence up to position t. The recursive steps are:

$$\alpha(s,t) = \sum_{s'} \alpha(s', t-1)\ p(s|s', o_t) \tag{4}$$

Hidden states are denoted as T_j, $j = 0..k-1$, the values $\alpha(T_j, t)$ in our focused crawling system are calculated as:

$$\alpha(T_j,t) = \sum_{j'=0}^{k-1} \alpha(T_{j'}, t-1)\ \frac{1}{z(o_t)} \exp(\sum_{i=1}^{m} \lambda_i f_i(T_j, T_{j'}, o_t))$$

$$z(o_t) = \sum_{j"=0}^{k-1} \exp \sum_{i=1}^{m} \lambda_i f_i(T_{j'}, T_{j"}, o_t)$$

`The Viterbi Algorithm` – the goal is to compute the most likely hidden state sequence given the data:

$$s^* = \arg\max_s p(s|o) \tag{5}$$

$\delta(s,t)$ is defined as the best score (i.e. the score with the highest probability) over all possible configurations of the state sequence ending in state s at position

t given the observations up to position t. That is

$$\delta(s,t) = \max_{s'} \delta(s', t-1)\ p(s|s', o_t) \tag{6}$$

This is the same recursive formulate as the forward values (Equation 4), except we replace sum with max.

Features and Feature Functions Each feature function $f(s,o,t)$ is defined as a factored representation:

$$f(s,o,t) = L(s_{t-1}, s_t, t) * O(o,t) \tag{7}$$

where $L(s_{t-1}, s_t, t)$ are transition feature functions, and $O(o,t)$ are observation feature functions.

1. **Edge Features:** Transition feature functions $L(s_{t-1}, s_t, t)$ can have two forms: L_1 and L_2. We use Edge feature $L_1(s_{t-1}, s_t, t)$ to capture the possible transitions from states s_{t-1} to s_t, and $L_2(s_{t-1}, s_t, t)$ to capture the possible states at position t.
   Formally, for all $i, j = 0, 1, ..., k-1$ so that specified $T_i, T_j \in S = \{T_{k-1}, T_{k-2}, ..., T_1, T_0\}$, we can have feature functions of the following form:

$$L_1^{(i,j)}(s_{t-1}, s_t, t) = \begin{cases} 1 \text{ if } s_{t-1} = T_i \text{ and } s_t = T_j \text{ is an allowed transition;} \\ 0 \text{ otherwise.} \end{cases}$$

$$L_2^{(i)}(s_{t-1}, s_t, t) = \begin{cases} 1 \text{ if } s_t = T_i \text{ exists;} \\ 0 \text{ otherwise.} \end{cases}$$

2. **Text Feature:** Maximal cosine similarity value between the content of a given candidate page and the set of targets. We define it as $O_1(o,t)$.
3. **Description Feature:** Cosine similarity value between the page description of a given candidate page and the target description. We define it as $O_2(o,t)$.
4. **Word Feature:** Word feature $O_w(o,t)$ identifies the keywords appearing in the page text, as described in sec. 3.3.

$$O_w(o,t) = \begin{cases} 1 \text{ if word } w \text{ appears in the current page at position } t; \\ 0 \text{ otherwise.} \end{cases}$$

   We may also use the count of word w as the value of this feature, instead of the binary value.
5. **URL Token Feature:** There are two possible kinds of URLs related to the current observed page: one is the URL of the current page itself, and another one is the URL the current page is pointing to. We define two token feature functions $O_3(o,t)$ and $O_4(o,t)$ to identify if the keywords appear in the URLs.

$$O_3(o,t) = \begin{cases} 1 \text{ if any of URLs in the current page at position } t \text{ contains} \\ \quad \text{at least one keyword;} \\ 0 \text{ otherwise.} \end{cases}$$

$$O_4(o,t) = \begin{cases} 1 \text{ if the URL of the current page at position } t\text{(contained in} \\ \quad \text{the parent page) contains at least one target keyword;} \\ 0 \text{ otherwise.} \end{cases}$$

6. Anchor Text Feature: We capture word w in the anchor surrounding text by defining two anchor features. At least 4 words are chosen from text around <a>...

$$O_{5,w}(o,t) = \begin{cases} 1 \text{ if word } w \text{ appears in the anchor text of the link in} \\ \quad \text{the parent page linking to current page at position } t; \\ 0 \text{ otherwise.} \end{cases}$$

$$O_{6,w}(o,t) = \begin{cases} 1 \text{ if word } w \text{ appears in the anchor text in the current} \\ \quad \text{page at position } t \text{ pointing to the page at position } t+1 \text{ ;} \\ 0 \text{ otherwise.} \end{cases}$$

4 Experiments

In this section, we conduct experiments to test our MEMM-based focused crawling approach empirically. The topics are chosen from the ODP categories, and the target pages are chosen based on the listed URLs under each category. The following table shows some information about the 10 topics for the experiments.

Topic	**# of Target Pages**	**# of training sequences**	**Start Urls**
Linux	19	11394	6
Biking	17	9790	2
Butterfly	17	12172	2
Hearthealthy	8	12928	2
Hockey	19	4368	2
Fitnessyoga	7	11680	3
Balletdance	7	12317	3
Skymaps	16	12073	3
Callforpapers	11	9632	3
Internetlaw	18	12210	4

4.1 Evaluation Methods

The *precision* is the percentage of the Web pages crawled that are relevant to the topic. The relevance assessment of a page p we are using is based on maximal cosine similarity to the set of target pages T compared with a confidence threshold γ. That is, if $\max_{d \in T} \cos(p,d) \geq \gamma$ then p is considered as relevant. Some topics may be sensitive to the threshold, therefore we choose the threshold values between 0.5-0.8 for general comparisons. Too high or too low threshold may result in too few or too many relevant pages based on the target pages and the start URLs, which does not provide sufficient information for comparison and for figure presentations.

The ability of the crawler to remain focused on the topical Web pages during crawling can also be measured by the average relevance of the downloaded documents[16–18]. In our system, since there are multiple pre-specified target pages, we used the *Maximum Average Similarity* σ.

$$\sigma = \max_{d \in T} \frac{\sum_{p \in S} \cos(p, d)}{|S|} \tag{8}$$

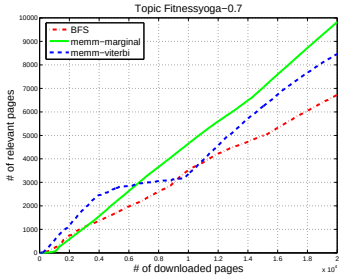
where T is the set of target pages, S is the set of pages crawled, $|S|$ is the number of targets.

4.2 Results

We have conducted two experiments. One is to compare MEMM-based method with different inference algorithms against Best-First Search(BFS) crawl. BFS crawl assigns priorities to all children of the current page using standard lexical cosine similarity between the content of the current page and target pages. The URL with the best score will be crawled first. Another one is to test the impact of the choice features on performance.

Comparison with Different Inference Algorithms: Viterbi Algorithm and Marginal Mode First we compare our MEMM-based methods with all the features against BFS crawl. We find that our MEMM-based crawl significantly outperforms BFS crawl on 8 out of 10 topics. The performance of three different crawling methods, BFS crawl, MEMM-marginal crawl, and MEMM-Viterbi crawl, on the topic *Fitnessyoga* is shown in Fig. 2 (a). All three methods give very good results on this topic, however, two MEMM-based crawls still work better than BFS crawl on the number of relevant pages returned, which also is confirmed on the *Maximum Average Similarity* metric, as shown in Fig. 2 (b).

[a]



[b]

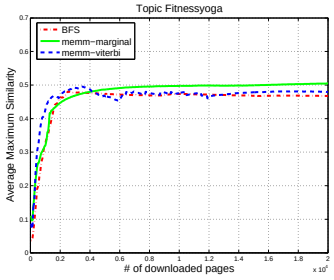
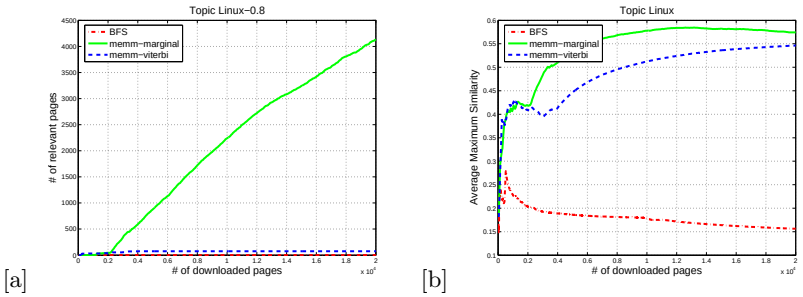


Fig. 2. Topic *Fitnessyoga*: (a) the number of relevant pages with threshold 0.7, (b) the Maximum Average Similarity.

The results on topic *Linux* are shown in Fig. 3 (a). Both MEMM-marginal crawl and MEMM-Viterbi crawl also outperform BFS crawl. MEMM with marginal mode shows significant improvement over BFS crawl, while MEMM with Viterbi



[a] [b]

Fig. 3. Topic *Linux*: (a) the number of relevant pages with threshold 0.8, (b) the Maximum Average Similarity.

algorithm shows slight improvement on the number of the relevant pages. However, MEMM-Viterbi crawl gives a very close performance to MEMM-marginal crawl on the *Maximum Average Similarity*, which significantly outperforms BFS crawl as shown in Fig. 3 (b). This shows that two MEMM-based methods stay on the topic, whereas BFS method crawls away from the topic resulting in poor performance.

Compared to the results only based on MEMM-marginal and MEMM-Viterbi crawls, we found that the marginal mode outperforms the Viterbi algorithm on 7 topics out of 10. In the focused crawling problem, finding the distribution for each individual hidden state at a particular instant is more important than finding the best "string" of hidden states of each Web page along the sequence, since there may be many very unlikely paths that lead to large marginal probability. To be more specific, let us see an example. If we have

"aaa" 30% probability
"abb" 20% probability
"bab" 25% probability
"bbb" 25% probability

In this example, "aaa" is the most likely sequence, 'a' is the most likely first character, 'a' is the most likely second character, 'b' is the most likely third character, but the string "aab" has 0 probability. Therefore, as we expected, MEMM-marginal crawl shows better performance than MEMM-Viterbi crawl in most of the cases in our experiments.

Comparison with Different Features In this section, we test the impact of the selected features on the performance. The last section demonstrated that MEMM-marginal crawl is better than MEMM-Viterbi crawl, so in this experiment, we choose to compare MEMM-marginal crawl with different features: with all features, with word feature only, and with the following features: Text feature, Description feature, URL token feature and Anchor text feature (See all features in Section 3.4). We denote them as *MEMM-marginal, MEMM-marginal-word, and MEMM-marginal-sim-meta-T* in the figures, respectively.

As the results show, MEMM-marginal crawl with all features performs consistently better than with Word feature only and with sim-meta-T features on

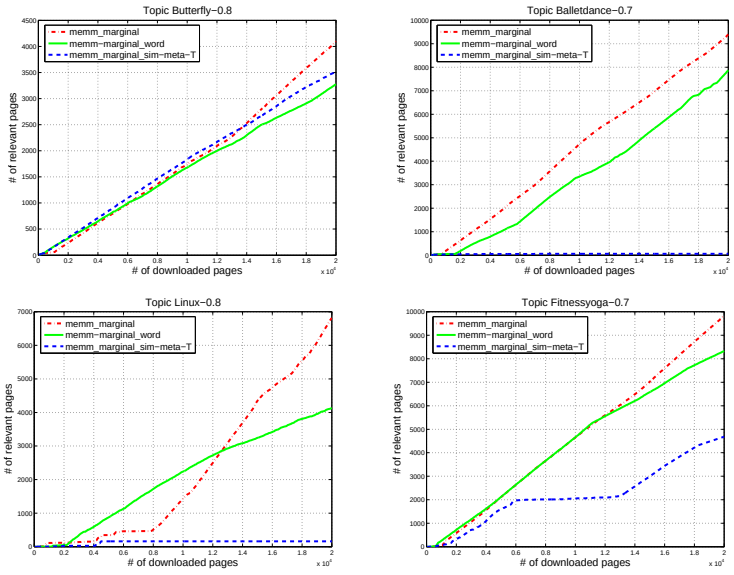


Fig. 4. Topic *Butterfly*, *Balletdance*, *Linux* and *Fitnessyoga*: Comparisons of different methods: with Word feature only, simmetaT feature only and the all features combination on the number of relevant pages within the set of downloaded pages.

almost all topics, except topic *Hearthealthy*, in which MEMM-marginal-word shows the best performance, and topic *Callforpapers*, in which MEMM-marginal-sim-meta-T shows the best. Fig. 4 shows some of the results. This confirms our approach that by using all the features, even if only some of them are present, relevant paths can be effectively identified.

5 Conclusion

Our approach is unique that we model the process of crawling by a walk along an underlying chain of hidden states, defined by hop distance from target pages, from which the actual topics of the documents are observed. When a new document is seen, prediction amounts to estimating the distance of this document from a target. In this way, good performance depends on powerful modeling of context as well as the current observations. The advantages and flexibility of MEMMs fit our approach well and are able to represent useful context. With Maximum Entropy Markov Models (MEMMs), we exploit multiple overlapping and correlated features, such as anchor text, to represent useful context and form a chain of local classifier models. We have studied the impact of different combination strategies, and the results showed that using marginal mode performs better than using Viterbi algorithm, and the crawler using the combination of all features performs consistently better than the crawler that depends on just one or some of them.

Acknowledgements This research was supported by NSERC, the MITACS Network of Centres of Excellence, and Genieknows.com. The input of Prof. Jeannette Janssen to this work is gratefully appreciated.

References

1. Cho, J., Garcia-Molina, H., Page, L.: Efficient Crawling through URL Ordering. In: Proceedings of the 7th World Wide Web Conference. (1998)
2. Chakrabarti, S., Punera, K., Subramanyam, M.: Accelerated Focused Crawling through Online Relevance Feedback. In: Proceedings of the 11th International WWW Conference. (1999)
3. Aggarwal, C., Al-Garawi, F., Yu, P.: Intelligent Crawling on the World Wide Web with Arbitrary Predicates. In: Proceedings of the 10th International WWW Conference. (2001)
4. Menczer, F., Belew, R.K.: Adaptive retrieval agents: Internalizing local context and scaling up to the Web. In: Machine Learning. (2000) 39(2/3):203–242
5. Johnson, J., Tsioutsiouliklis, K., Giles, C.L.: Evolving Strategies for Focused Web Crawling. In: Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003). (2003)
6. Ehrig, M., Maedche, A.: Ontology-focused crawling of web documents. In: SAC '03: Proceedings of the 2003 ACM symposium on Applied computing, New York, NY, USA, ACM (2003) 1174–1178
7. Pant, G., Srinivasan, P.: Learning to Crawl: Comparing Classification Schemes. In: ACM Trans. Information Systems. (2005) vol. 23, no. 4
8. Pant, G., Srinivasan, P.: Link Contexts in Classifier-Guided Topical Crawlers. IEEE Transactions on Knowledge and Data Engineering **18**(1) (2006) 107–122
9. Frnkranz, J.: Hyperlink ensembles: A case study in hypertext classification. Information Fusion **3**(4) (2002) 299–312
10. Rennie, J., McCallum, A.: Using Reinforcement Learning to Spider the Web Efficiently. In: Proceedings of the Sixteenth International Conference on Machine Learning(ICML-99). (1999)
11. Diligenti, M., Coetzee, F., Lawrence, S., Giles, C., Gori, M.: Focused Crawling Using Context Graphs. In: Proceedings of the 26th International Conference on Very Large Databases (VLDB 2000). (2000)
12. Liu, H., Janssen, J., Milios, E.: Using hmm to learn user browsing patterns for focused web crawling. Data & Knowledge Engineering **59**(2) (2006) 270–291
13. McCallum, A., D.Freitag, Pereira, F.: Maxiumu Entropy Markov Models for Information Extraction and Segmantation. In: Proceedings of the Seventeenth International Conference on Machine Learning. (2000) 591–598
14. Nocedal, J., S.J.Wright: Numerical Optimization. Springer (1999)
15. Sha, F., Pereira, F.: Shallow Parsing with Conditional Random Fields. In: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology. (2003) 134–141
16. Menczer, F., Pant, G., Srinivasan, P., Ruiz, M.: Evaluating Topic-Driven Web Crawlers. In: Proceedings of the 24th ACM/SIGIR Conference. Research and Development in Information Retrieval. (2001)
17. Menczer, F., Pant, G., Srinivasan, P.: Topical Web Crawlers: Evaluating Adaptive Algorithms. ACM TOIT **4**(4) (2004) 378–419
18. Srinivasan, P., Menczer, F., Pant, G.: A General Evaluation Framework for Topical Crawlers. Information Retrieval **8**(3) (2005) 417–447