Focused Crawling

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Engineering

IN

COMPUTER SCIENCE AND ENGINEERING

by

Babaria Rashmin N.

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

July 2007

To My Beloved Parents

Nandlalbhai, Manishaben

Acknowledgments

It is a pleasure to thank all those who made this thesis possible.

I would like to thank my guide Dr. Chiranjib Bhattachryya for giving me the opportunity to work with him. Thank you sir, for your valuable guidance, for the patience with which you taught me, for your support and encouragement.

I extend my thanks to Prof. M. N. Murty for his valuable suggestions during my project work.

I would like to thank Krishnan Suresh, Sivaramakrishnan Kaveri and J. Saketha Nath for helping me in my project.

I would also like to thank all my ML lab-mates for providing a stimulating and fun environment for work. Life without friends in IISc would have been difficult and dull, so I like to thank all my friends in IISc especially Mehul, Tarun, Megha and Prachee. Thanks, to all those whom I have not mentioned here,but have helped me in some or the other way in completing this work.

Lastly but most importantly, I would like to thank my parents and my sister for having faith and confidence in me, for encouraging and supporting me.

Publications based on this Thesis

1. Focused Crawling with Scalable Ordinal Regression Solvers. R. Babaria, J. Saketha Nath, Krishnan S, Sivaramakrishnan K R, C. Bhattacharyya, M. N. Murty. In proceedings of the ICML-2007 conference

Abstract

Focused crawling is an efficient mechanism for discovering resources of interest on the web. Link structure is an important property of the web that defines its content. In this thesis, FOCUS a novel focused crawler is described, which primarily uses the link structure of the web in its crawling strategy. It uses currently available search engine APIs, provided by Google, to construct a layered web graph. This layered model of the web is used to learn the link that leads to topic pages. It is formulated as an ordinal regression problem that when solved gives link distance of topic pages from any given page. This directly produces an ordering among the links to be crawled. The ordinal nature of the problem removes any need for a negative class which is a problem with the existing crawlers. The large scale nature of the web poses scaling problems to current ordinal regression solvers. To overcome this a novel large scale clustering based ordinal regression solver using Second Order Cone Programming(SOCP) is proposed. Proposed approach is implemented on top of nutch open source crawler, which uses wellknown MapReduce distributed model to make the crawler scalable. Experimental results on different datasets show that the proposed method is competitive.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, the World Wide Web has become the primary medium for personal, social and commercial information dissemination. Finding useful information from the web, which is a large scale distributed structure, require efficient search strategies. Most search engines maintain an index of web pages which is exploited to provide up-to-date information related to user queries. Highly distributed and dynamic nature of the source of web content is a major source of practical problems for search engines to maintain up-to-date index of the web content as they have to crawl the web periodically. Traversal/crawling of the whole web takes order of days/weeks, large network bandwidth and storage capacity. The delay in updation of index causes display of irrelevant pages and/or missing highly relevant pages to the user queries. The World Wide Web is growing at a faster rate compared to processor speed, network bandwidth and storage capacity which requires efficient scalable solutions.

Focused crawler is an automated mechanism to efficiently find pages relevant to a topic on the web. Focused crawlers were proposed to traverse and retrieve only a part of the web that is relevant to a particular topic, starting from a set of pages usually referred to as the seed set. It makes efficient usage of network bandwidth and storage capacity. Focused crawling provides a viable mechanism for frequent updation of search engine indexes. They have been useful for other applications like distributed processing of the complete web, with each crawler assigned to a small part of the web. They have also been used to provide customized alerts, personalized/community search engines, web databases and business intelligence.

Focused crawler is a tool used for topic specific information discovery on the web. Many variants of focused crawlers have been proposed in the literature. They generally differ in the strategy used for the crawl. A major problem associated with these crawlers is they require a negative set to compute a classifier. We propose an ordinal regression based approach that does not require such a set. We generate a layered web graph around the seed set using search APIs provided by Google. We formulate finding the link distance from a given page to topic page as an ordinal regression problem. The solution to the problem provides priority/relevance score of any page during the crawl, effectively defining the crawl strategy. We provide a clustering based approach to ordinal regression for handling large scale data generally found on the web, which is extension of the clustering based classification formulation [1].

We first implemented focused crawling on top of Nalanda iVia focused crawler [2] to compare results between our approach and state of the art approach. Nalanda crawler is made to run on a single computer, but generally focused crawler should run on distributed system. Nutch is general crawler, which uses googles's MapReduce programming model for scalability. Programs, which use MapReduce model run efficiently on distributed system. We implemented focused crawler on top of nutch software, which can be run on cluster of machines and is scalable.

The outline of the project report is as follows, Chapter 2 gives a brief overview of current focused crawling technology. Chapter 3 describes FOCUS: a novel focused crawling formulation as ordinal regression. Overview of ordinal regression and clustering based large scale ordinal regression formulation are described in chapter 4. Chapter 5 details the experiments and discusses the results in comparison with other crawlers. We developed an opensource focused crawling code on top of Nutch. Details are described in chapter 6. Chapter 7 concludes and describes improvements for future.

Chapter 2

Focused Crawling : A review

The World Wide Web is an instance of a large graph. Different inherent properties of the web graph have been used extensively in deriving algorithms like PageRank[3] and finding hubs and authoritative sources[4] which are useful for searching the web. Its structure provides an effective means of finding information about a particular topic. An interesting question is starting from a given a set of pages, that define a certain topic, is it possible to efficiently discover more pages of the same topic. This method of finding on-topic pages is called focused crawling[5]. Crawlers generally have one to three components. One of the components is the crawler that downloads the pages given Universal Resource Locators(URLs) in a priority queue, processes the pages for URLs, constructing the link graph, etc. The other optional components provide the strategy/priority to the URLs to be downloaded by the crawler component. Each of the proposed focused crawlers differ mostly in the strategy used to crawl the links. Early methods of topic based crawling include Fish-Search[6], Shark-Search[7] and PageRank based topic search[8].

2.1 Focused crawling

Focused crawling was coined by [9] as an efficient resource discovery system. It has three components – crawler, classifier and distiller. As shown in Figure 2.1, existing document taxonomy is used to define the topics of interest and irrelevant topics. User provides seed

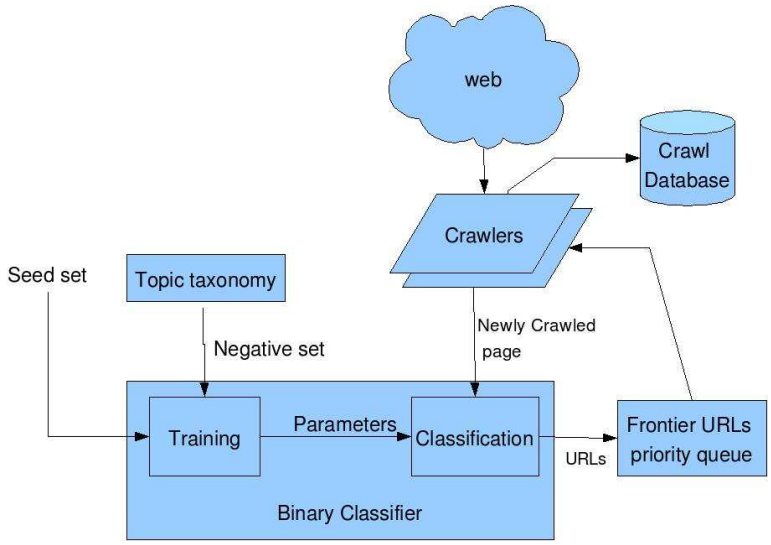

Figure 2.1: Focused crawling architecture

documents for nodes in the taxonomy tree in which he is interested in. This allows the user to specify interest in several unrelated topics. Negative set is selected randomly from the irrelevant nodes in the taxonomy tree. For example (figure 2.2)if user is interested in NASCAR then negative set may contain web pages from other car racing nodes, other sports nodes and rest of the topic taxonomy tree. Classifiers are learnt at each internal node of the tree, which gives the probability of the document belonging to the node. Urls in positive seed set are first inserted in priority queue. Crawler thread pics best score url from the priority queue and fetches the web page pointed to by that url. The web page is given to the classifier module to find out relevance score of that page. If that page is relevant then all the outlinks in that page are inserted in the priority queue with priority proportional to the relevance score of the web page. The distiller periodically runs through the crawled pages to find the hubs and authorities[4] to prioritize URLs that are found to be in hubs. The priority assigned is to a document and hence all URLs in the document have equal priority in the crawl.

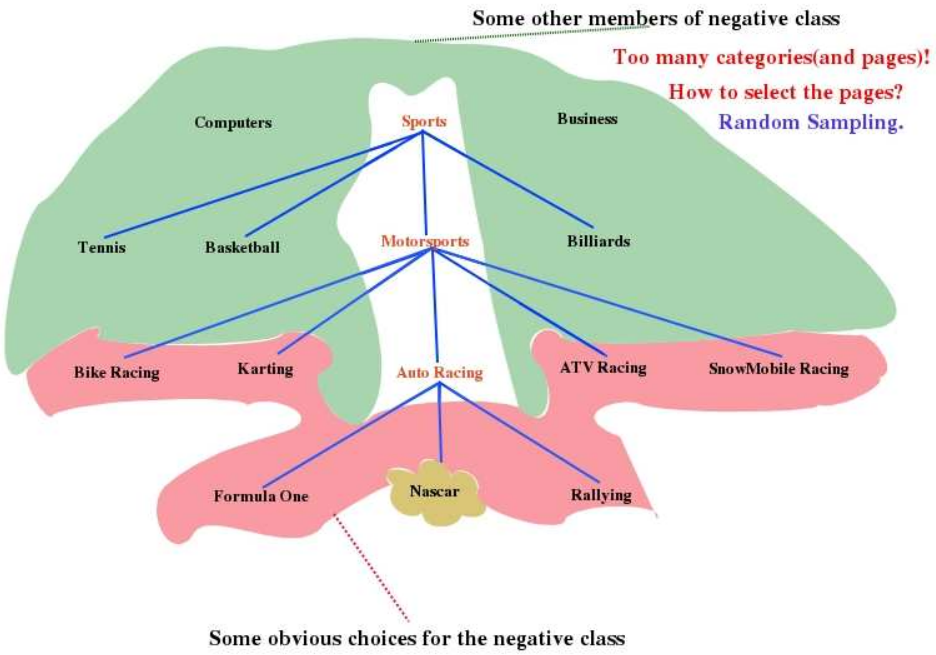

Figure 2.2: Part of the topic taxonomy tree — describes sports hierarchy

2.2 Accelerated focused crawling

An improved version was proposed in [10] which extends the previous baseline focused crawler to prioritize the URLs within a document. Relevance of a crawled page is obtained using the document taxonomy as explained above using the 'baseline' classifier. The URLs within the document are given priority based on the local neighborhood of the URL in the document. An apprentice classifier learns to prioritize the links in the document. Once sufficient number of source pages and the target pages pointed to by the URL in the source page are downloaded and labeled as relevant/irrelevant, the apprentice is learnt. A representation for each URL based on target page relevance, source page relevance, Document Object Model(DOM) structure, co-citation and other local source page information is constructed. The apprentice is trained online to predict the relevance of the target page pointed to by a URL in the source page. The relevance so calculated is used to prioritize the URLs. The apprentice is periodically

retrained to improve the performance. Both these methods depend on the usage of quality document taxonomy for good performance. The dependence on the document taxonomy makes it inapplicable to applications where the topic is too specific.

2.3 Intelligent crawling

Intelligent crawling was proposed in [11] that allows users to specify arbitrary predicates. It suggests use of arbitrary implementable predicates that use four sets of document statistics including source page content, URL tokens, linkage locality and sibling locality in the classifier to calculate the relevance of the document. The source page content allows prioritizing different URLs differently. URL tokens help in getting approximate semantics about the page. Linkage locality is based on the assumption that web pages on a given topic are more likely to link to those of the same topic. Sibling locality is based on the assumption that if a web page points to pages of a given topic then it is more likely to point to other pages on the same topic. The focused crawler in [9] uses only linkage locality and sibling locality properties which may not be applicable to all topics. It proposes to vary the dependence on each set of features online by reevaluating over currently crawled pages. The intelligent crawler, given a set of seed pages, initially crawls randomly to get a set of examples to train its classifier and learns the importance of difference feature sets. Each statistic changes based on the actual visiting of the page. It allows intelligent crawling adapt itself in order to maximize the performance. The reinforcement based learning of the classifier allows it to adapt for different topics and seed set.

2.4 Focused crawling using context graph

Focused crawlers proposed in [9, 10, 11] use a greedy approach to crawl the web. They forego overall crawl performance for instantaneous gains. It proposes use of context graph of the web to calculate the relevance of target page of the link to the topic. The problem faced by most focused crawlers based on a greedy strategy is that some highly relevant documents with paths

through irrelevant documents are not crawled. Focused crawler in [12] proposes to solve the credit assignment problem by constructing a layered graph model of the web. It uses already existing web search engines to construct (by 'backward crawling') a deep layered graph structure, using back references, which when crawled lead to the given set of seed pages. Learning a single classifier that maps a document into a set of r+2 classes corresponding to 0,1,$\cdots$,r and a category "other" poses difficulty due to the absence of a good model or training set for the "other" category. A modified naive bayes classifier is constructed that allows a document to be assigned to the layer with maximum probability. The relevance of the document to that layer is calculated and the document is discarded as an outlier/irrelevant if the relevance is less than a threshold. Documents classified into a layer nearest to the seed pages are given preference in the crawl.

Instead of using a search engine to learn the context graph [13] use topic-specific browsing of an user to define the back-links and the topic pages. Interesting pages are marked as topic pages during the browsing of the user. This allows for arbitrary group of topics to be given for focused crawling. They group all the documents into 5 clusters using K-Means and apply EM algorithm to optimize the quality of these 5 clusters. They propose HMM based statistical method to estimate the likelihood of pages leading to a target topic directly or indirectly. Given this model and its parameters, the task is to find the most likely state sequence given the observed web page sequence. Determining this sequence is efficiently performed by Viterbi algorithm.

2.5 Evaluation

The aim of any focused crawler is to start from a set of pages relevant to a given topic and traverse specific links to collect pages about the topic without fetching pages unrelated to the topic. When an uncrawled URL x is found in page y, the focused crawler adds x to the queue with relevance estimate based on y. When the page x is actually fetched the actual relevance can be found. The fraction of relevant pages fetched is called the harvest rate[10]. Let N be the number of pages crawled that have been crawled by any given time t, then the harvest rate

at time t is defined as the fraction of crawled pages that are relevant. More precisely

$$Harvest\ rate = \frac{1}{N}\sum_{i=1}^{N} I_{(\rho_i > \theta)} \tag{2.1}$$

where ρ_i is the relevance of the i th crawled page and $I_{(\rho_i > \theta)}$ is an indicator function (it is 1 if $\rho_i > \theta$, otherwise it is 0). The function ρ assigns a value in $[0, 1]$ for every page.

Chapter 3

FOCUS: a new look at focused crawling

3.1 Explore link structure

Current methods for focused crawling prioritize the links to be crawled based on the probability/relevance scores. They also depend on having a high quality document taxonomy. Many depend on the existence of a negative set of pages. These resource requirements of current focused crawlers is mainly based on using a classifier based design. It has been observed in [14, 15] that document d is semantically closer to documents, hyperlinked with document d, than to documents, not hyperlinked with document d. Thus pages which are one link away are more semantically closer to seed pages than pages that are two to three links away. We propose to rank the documents directly based on their link distance to the topic pages rather than find the likelihood/relevance scores. The ranking directly creates an ordering among the documents of the web due to its link structure.

Learning the link distance of a given page to the topic page requires knowing a lot more than the seed pages. We need examples of pages which are at some link distance to the topic pages. Given that we have the seed set of pages we need pages that lead to these pages. We model the whole web as a layered graph with the pages relevant to the seed pages/topic forming the first layer. Similarly, pages which have links to topic pages forming the second layer, pages having links to these second layer pages forming the third layer and so on. Given this layered graph model of the web, the strategy is to crawl links that lead to topic pages through one link

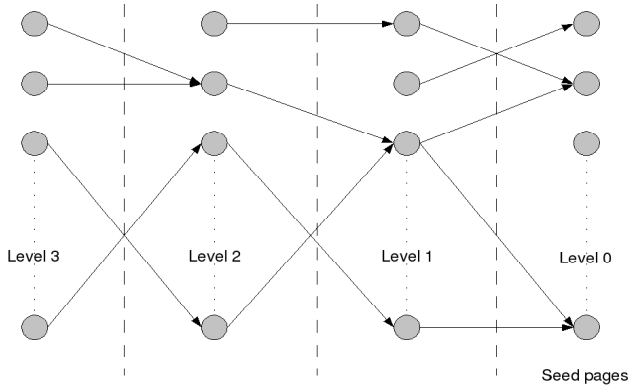

Figure 3.1: Web Graph formed from the seed pages shows approximately the local structure around the seed pages. The levels are constructed in the reverse order, from seed pages to level 3 pages, using the back-links.

compared to links that lead to topic pages through two links, i.e, we would like to traverse links in layer 2 than layer 3 than layer 4 and so on. This inherently gives an ordering/ranking of the links to be crawled. The learning of the different layers of this web graph allows us to learn the kind of topics that lead to the topic pages. For example, consider the case where the topic is *graduate courses*. The course web pages can be reached from either the university pages or from the department pages to instructors pages or through students pages. Most crawlers would not crawl pages of universities and departments to find the course pages. When the layered web graph around the course pages is formed other courses, students and instructors pages form the level 1 pages. The department pages might form the level 2 pages and the university pages might form the level 3 pages. When this structure of the web around the course pages is learnt it is easier to find pages of other courses.

3.2 Focused crawling as ordinal regression

The layered web graph as in Fig.3.1 is constructed to learn about pages that lead to the target pages. We use the seed set to find the back-links, the pages that point to these seed pages. This information is not directly available from the seed pages. We use existing general search engine API - GOOGLE SOAP Search API[16] to get the back-references to the seed pages.

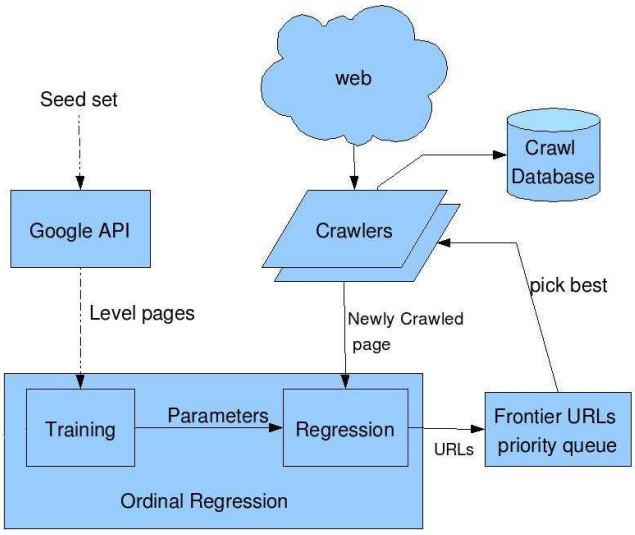

Figure 3.2: Block diagram of FOCUS

These pages form examples of layer two. We get the back-links to these layer two pages to create examples of layer three. We create a layered graph of depth 'n'. Given these set of pages we would like to learn a function that takes a document representation and accurately computes the layer of a given page/document. This general type of ranking problems are solved using ordinal regression in Machine Learning. We formulate focused crawling strategy as an ordinal regression problem.

This approach is similar to [12, 13] where a context graph is created. In both the approaches layer information is being used to create a multi-class problem. The layered graph provides a way to characterize the pages, when crawled, are more likely to lead to target pages. Our strategy is to crawl pages such that we get as many target pages as possible in least amount of time. This requires crawling links that are at lesser link distance from the seed pages. The use of ordinal regression to directly learn a function that ranks the documents differs from using multiple classifiers employed by all the current focused crawlers. Since ordinal regression has only been studied recently in Machine Learning, practically applicable solutions are available only in the recent years. Ordinal regression has not been used by any current focused crawlers.

3.3 FOCUS architecture

As shown in figure 3.2 seed set is collected from the user. As a preprocessing GOOGLE SOAP Search API[16] is used to find level information. Web pages pointed to by URLs in all the levels are fetched and ordinal regressior is learnt using those web pages. URLs in the seed set are inserted into priority queue as starting point of the actual crawling. The crawler takes a best URL from the priority queue to download and process the web document. Ordinal regressior is used to predict level of the newly fetched page and all the outlinks in that page are inserted into priority queue with priority proportional to the level of the page. Thus, the ordinal regressor creates ordering among the URLs for the crawl, allowing pages closer to the topic pages be crawled before pages that are less related to the topic.

Chapter 4

Clustering based large scale ordinal regression

4.1 Ordinal regression

In the following we provide a brief overview of the Ordinal Regression(OR) problem. The problem of OR can be understood as building predictive models such that it will give a label y given a data point x. The label y takes values in a finite set and the data point is described by a point in a finite dimensional vector space. Given a data set

$$D = \{({x_i}^j, y_i) \mid i = 1, \cdots, r,\ j = 1, \cdots, n_i\} \tag{4.1}$$

where r is the number of labels, n_i is the number of data points having label i and $n = \sum_{i=1}^{r} n_i$ is the total number of data points. The key problem is to compute a function $f : \mathbb{R}^m \rightarrow \{1, \ldots, r\}$. Such formulations find ready appeal in many problems such as ranking [17].

An $O(n^3)$ formulation was proposed in [18] which maps the data onto $\mathbb{R}$, the real line as shown in Fig.4.1, and divides it into *r* regions such that points of rank *k* fall in k^{th} region. It uses a linear function $w^\top x,\ w \in \mathbb{R}^m,\ x \in \mathbb{R}^m$ to map the points to $\mathbb{R}$. It uses a perceptron like online algorithm to find the parameters $w \in \mathbb{R}^m$ and thresholds $b_i \in \mathbb{R},\ i = 1, \cdots, r-1$ that

define the boundaries of the regions. A given point x has rank *k* if

$$w^\top x + b_i > 0,\ \forall i < k$$

and

$$w^\top x + b_i \leq 0,\ \forall i \geq k$$

where b_0 is $-\infty$ and b_r is ∞.

In [19] a large margin formulation was proposed. It is given by

$$\begin{aligned}
\min_{(w,b_1,...,b_{r-1})} \quad & \frac{1}{2}w^\top w + \sum_{i=1}^{r}\sum_{j=1}^{n_i} \xi_i^j + \xi_i^{*j} \\
s.t. \quad & w^\top x_i^j + b_i \leq -1 + \xi_i^j, \quad i = 1,2,...,r-1, \quad j = 1,2,...,n_i \\
& w^\top x_i^j + b_{i-1} \geq 1 - \xi_i^{*j}, \quad i = 2,3,...,r, \quad j = 1,2,...,n_i \\
& \xi_i^j \geq 0, \quad i = 1,2,...,r-1 \\
& \xi_i^{*j} \geq 0, \quad i = 2,3,...,r
\end{aligned} \tag{4.2}$$

that uses SVM to solve ordinal regression problem for w and b was proposed. Here the margin between the closest ranks given by $\frac{2}{\|w\|_2}$ is maximized while minimizing the errors given by ξ_i^j and ξ_i^{*j}. The Large margin approach is used in many learning theory problems. A Gaussian process formulation of ordinal regression was outlined in [20].

An improved large margin formulation was given in [21]. It also provides a SMO type algorithm for solving the ordinal regression problem efficiently. The formulation in [21] can

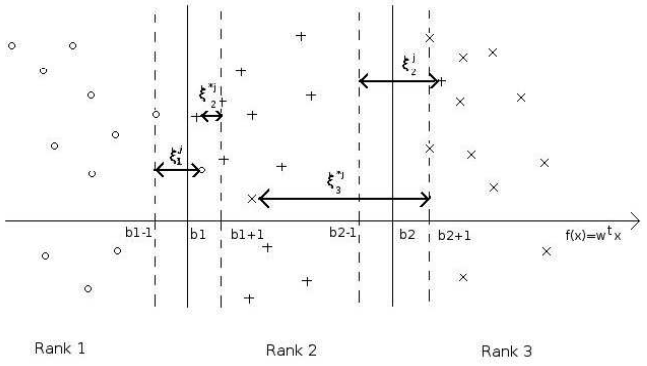

Figure 4.1: An illustration of the slack variables ξ_i^j and ξ_i^{*j}. Data points from different ranks are represented by circles, plus and cross. Data points are mapped to the line by the function $f(x) = w^\top x$

be written as

$$
\begin{aligned}
\min_{(w,b_1,\ldots,b_{r-1})} \quad & \frac{1}{2}w^\top w + \sum_{i=1}^{r}\sum_{j=1}^{n_i} \xi_i^j + \xi_i^{*j} \\
s.t. \quad & w^\top x_i^j + b_i \leq -1 + \xi_i^j, \quad i = 1,2,...,r-1, \quad j = 1,2,...,n_i \\
& w^\top x_i^j + b_{i-1} \geq 1 - \xi_i^{*j}, \quad i = 2,3,...,r, \quad j = 1,2,...,n_i \\
& \xi_i^j \geq 0, \quad i = 1,2,...,r-1 \\
& \xi_i^{*j} \geq 0, \quad i = 2,3,...,r \\
& b_i - b_{i-1} > 0, \quad i = 2,3,...,r-1
\end{aligned} \tag{4.3}
$$

which explicitly puts the constraints on b_i's. Minimizing $w^\top w$ is equivalent to bounding the

L_2 norm of w. Above formulation is equivalent to

$$\begin{aligned}
\min_{(w,b_1,\ldots,b_{r-1})} \quad & \sum_{i=1}^{r}\sum_{j=1}^{n_i} \xi_i^j + \xi_i^{*j} \\
s.t. \quad & w^\top x_i^j + b_i \leq -1 + \xi_i^j, \quad i = 1,2,...,r-1, \quad j = 1,2,...,n_i \\
& w^\top x_i^j + b_{i-1} \geq 1 - \xi_i^{*j}, \quad i = 2,3,...,r, \quad j = 1,2,...,n_i \\
& \|w\|_2 \leq W \\
& \xi_i^j \geq 0, \quad i = 1,2,...,r-1 \\
& \xi_i^{*j} \geq 0, \quad i = 2,3,...,r \\
& b_i - b_{i-1} > 0, \quad i = 2,3,...,r-1
\end{aligned} \tag{4.4}$$

This formulation of ordinal regression is dependent on the problem/data size. Due to the number of pages that need to be considered in the case of focused crawling current methods are not sufficiently efficient, we propose a clustering based large scale ordinal regression formulation that is independent of problem size.

4.2 Clustering based ordinal regression

Let Z_i be the random variable that generates the data points of rank 'i'. Assume that the distributions of Z_i can be modeled using mixture models. Let k_i be the number of components of Z_i where each component distribution has spherical covariance. Let $X_i^j,\ j = 1,\cdots,k_i$ be the random variable generating the j^{th} component of Z_i whose second order moments are given by $(\mu_i^j, {\sigma_i^j}^2 I)$. Then the probability density function of Z_i can be written as

$$f_{Z_i}(\mathbf{z}) = \sum_{j=1}^{k_i} \rho_i^j f_{X_i^j}(\mathbf{z}),$$

where, ρ_i^j are the mixing probabilities ($\rho_i^j \geq 0$, $\sum_{j=1}^{j=k_i} \rho_i^j = 1$). Any good clustering algorithm will correctly estimate the second order moments of the components. BIRCH is one such clustering algorithm, that scales well for large datasets. Given these estimates of second order

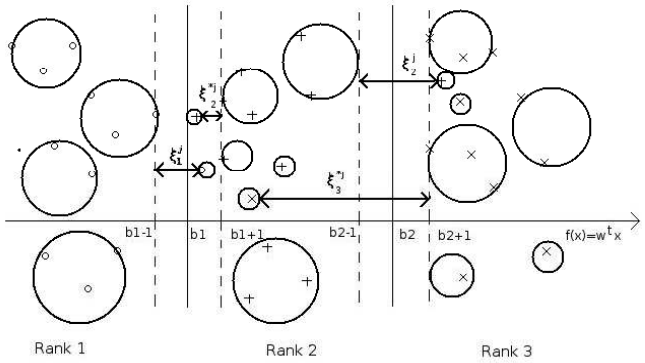

Figure 4.2: An illustration of the slack variables ξ_i^j and ξ_i^{*j}. Clusters from different ranks are represented by circles. The number of Clusters is sufficiently low compared to number of Data points.

moments, an optimal regressor that generalizes well can be built.

The constraints $w^\top x_i^j + b_i \leq -1 + \xi_i$ and $w^\top x_i^j + b_{i-1} \geq 1 - \xi_i^*$ ensure that these samples are classified correctly with high probability. We can as well write these constraints as $w^\top Z_i + b_i \leq -1 + \xi_i$ and $w^\top Z_i + b_{i-1} \geq 1 - \xi_i^*$. Since Z_i's are random variables, the constraints cannot be always satisfied. Thus, we ensure that with high probability, the events $w^\top Z_i + b_i \leq -1 + \xi_i$ and $w^\top Z_i + b_{i-1} \geq 1 - \xi_i^*$ occur.

$$\begin{aligned} P(w^\top Z_i + b_i \leq -1 + \xi_i^j) &\geq \eta \\ P(w^\top Z_i + b_{i-1} \geq 1 - \xi_i^{*j}) &\geq \eta \end{aligned} \tag{4.5}$$

where η is user defined parameter. η lower bounds the classification accuracy. Since the distribution of Z_i is a mixture model, in order to satisfy (4.5), it is sufficient that each of its components satisfy these constraints,

$$\begin{aligned} P(w^\top X_i^j + b_i \leq -1 + \xi_i^j) &\geq \eta \\ P(w^\top X_i^j + b_{i-1} \geq 1 - \xi_i^{*j}) &\geq \eta \\ \forall i = 1, \cdots, r & \\ \forall j = 1, \cdots, k_i & \end{aligned} \tag{4.6}$$

which leads to the following formulation of ordinal regression

$$\begin{aligned}
\min_{(w,b_1,...,b_{r-1})} \quad & \sum_{i=1}^{r}\sum_{j=1}^{n_i} \xi_i^j + \xi_i^{*j} \\
s.t. \quad & P(w^\top X_i^j + b_i \leq -1 + \xi_i^j) \geq \eta, \quad i = 1,2,...,r-1, \quad j = 1,2,...,k_i \\
& P(w^\top X_i^j + b_{i-1} \geq 1 - \xi_i^{*j}) \geq \eta, \quad i = 2,3,...,r, \quad j = 1,2,...,k_i \\
& \|w\| \leq W \\
& \xi_i^j \geq 0, \qquad i = 1,2,...,r-1 \\
& \xi_i^{*j} \geq 0, \qquad i = 2,3,...,r \\
& b_i - b_{i-1} > 0, \qquad i = 2,3,...,r-1
\end{aligned} \tag{4.7}$$

The constraints in the optimization problem (4.7) are probabilistic. In order to solve the optimization problem (4.7), the constraints need to be written as deterministic constraints. We use the multivariate generalization of Chebyshev-Cantelli inequality [1] to derive deterministic constraints.

$$Prob(X \in \mathcal{H}) \geq \frac{s^2}{s^2 + w^\top \Sigma w} \tag{4.8}$$

where $s = (-b - w^\top \mu)_+,\ (x)_+ = max(x, 0)$. Applying (4.8), the constraints for rank 'i' can be handled by setting(note that $\Sigma = \sigma_i^{j2} I$)

$$P(w^\top X_i^j + b_{i-1} \geq 1 - \xi_i^{*j}) \quad \geq \frac{(w^\top \mu_i^j + b_{i-1} - 1 + \xi_i^{*j})_+^2}{(w^\top \mu_i^j + b_{i-1} - 1 + \xi_i^{*j})_+^2 + \sigma_i^{j2} w^T w} \quad \geq \eta$$

which results in the constraint

$$w^\top \mu_i^j + b_{i-1} \geq 1 - \xi_i^{*j} + \kappa\, \sigma_i^j \|w\|_2 \tag{4.9}$$

where, $\kappa = \sqrt{\frac{\eta}{1-\eta}}$. Similarly the constraint for

$$P(w^\top X_i^j + b_i \leq -1 + \xi_i^j) \geq \eta$$

can be derived as

$$w^\top \mu_i^j + b_i \leq -1 + \xi_i^j - \kappa\, \sigma_i^j \|w\|_2 \tag{4.10}$$

Substituting (4.9) and (4.10) in equation (4.7) produces the following deterministic optimization problem:

$$\begin{aligned}
\min_{(w,b_1,\ldots,b_{r-1})} \quad & \sum_{i=1}^{r}\sum_{j=1}^{n_i} \xi_i^j + \xi_i^{*j} \\
s.t. \quad & w^\top \mu_i^j + b_i \leq -1 + \xi_i^j - \kappa\, \sigma_i^j W \quad i = 1,2,...,r-1, \quad j = 1,2,...,k_i \\
& w^\top \mu_i^j + b_{i-1} \geq 1 - \xi_i^{*j} + \kappa\, \sigma_i^j W \quad i = 2,...,r, \quad j = 1,2,...,k_i \\
& \|w\|_2 \leq W, \;\; \xi_i^j \geq 0 \quad \xi_i^{*j} \geq 0 \quad \forall i,j \\
& b_i - b_{i-1} > 0 \quad i = 1,...,r-1
\end{aligned} \tag{4.11}$$

The ordinal regression formulation (4.11) is an SOCP problem. This problem can be solved to obtain the optimal values of w and b. Note that the k_i is number of constraints for each rank 'i' in (4.11) compared to n_i in (4.3) as shown in Fig.4.2.

We first cluster data points using any scalable clustering algorithm like BIRCH[22], which provides the second order moments of all the clusters. Now we solve the SOCP problem (4.11) using solvers like SEDUMI[23].

Chapter 5

Experiments

The chapter consists of 2 parts: The first part discusses the quality of the pages fetched and the second part looks at the performance and scalability of the clustering based OR algorithm.

5.1 FOCUS

The purpose of these experiments is to study the performance of the crawler on topics which are difficult to crawl, where difficulty is measured in terms of the harvest rate of the *baseline crawler*[9, 10]. Nalanda iVia focused crawler[2] is used as the baseline crawler; it implements the focused crawler in [9] with a logistic regression based binary classifier. The *apprentice*[10] is not employed in the experiments. But this does not affect the conclusions, as the apprentice is orthogonal to any crawler and will only lead to performance improvement.

The topics chosen are Mutual Funds, NASCAR, Soccer, and Cancer. Mutual Funds pages are heavily interlinked with pages on investment. This was observed[9] to cause difficulties when crawling. NASCAR pages seem very similar to pages on other motorsports organizations/races(e.g. INDY 500). This is due to the fact that a lot of the pages have similar content including terminology, names, racing circuits etc.

Preparation of the training set: A set of seed pages is first collected for each topic, from sources like Wikipedia and links returned by queries to general purpose search engines. Then the layered graph is created using google api [16]. This graph consisted of 5 layers. The

Table 5.1: Datasets: Categories and training set sizes

Category	**Seed**	**1**	**2**	**3**	**4**
NASCAR	1705	1944	1747	1464	1177
Soccer	119	750	1109	1542	3149
Cancer	138	760	895	858	660
Mutual Funds	371	395	540	813	1059

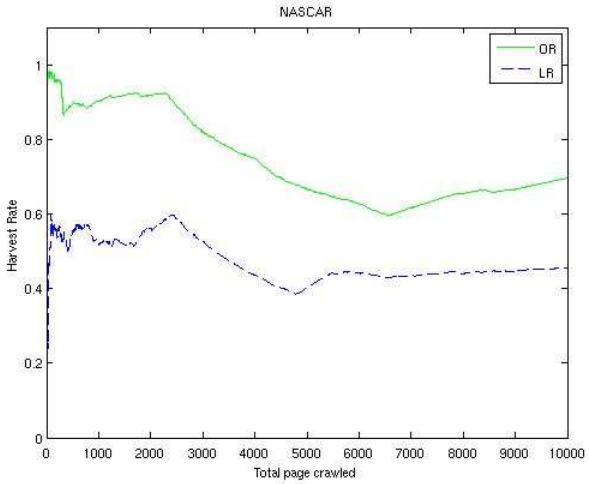

Figure 5.1: NASCAR harvest rate

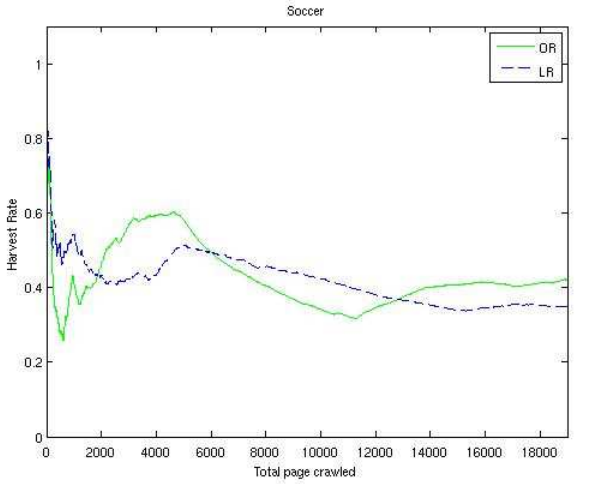

Figure 5.2: Soccer harvest rate

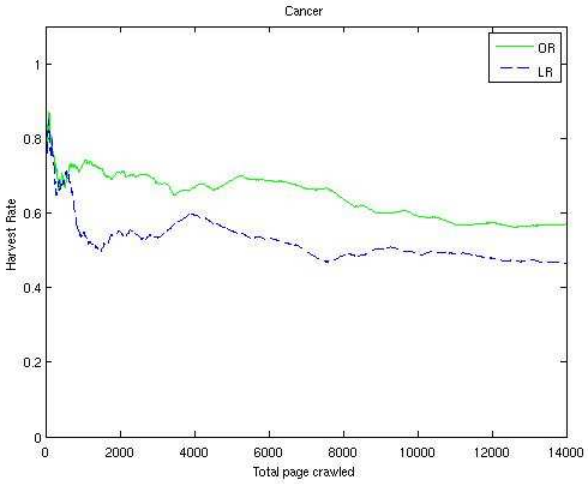

Figure 5.3: Cancer harvest rate

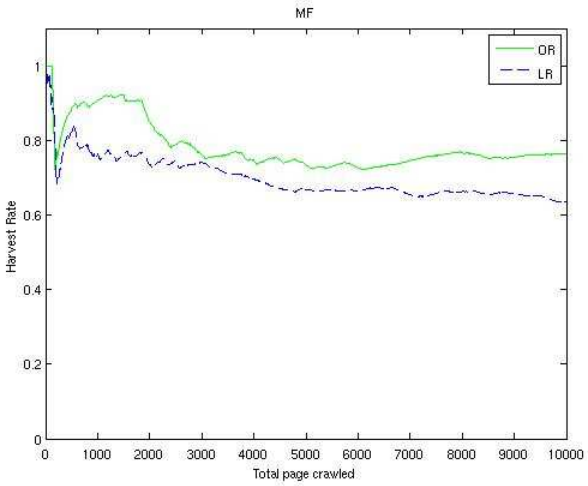

Figure 5.4: Mutual fund harvest rate

Table 5.2: #Good(#Bad) is the number of relevant (irrelevant) web pages crawled by baseline crawler. It indicates the difficulty in crawling these categories. Last two columns show the harvest rate.

Dataset	#Good/#Bad	**Baseline**	**OR**
NASCAR	11530/19646	.3698	.6977
Soccer	10167/9131	.34	.4952
Cancer	6616/12397	.4714	.58
Mutual Fund	9960/10992	.526	.5969

number of pages that are collected through back-links vary for different levels and categories. This is done to test the robustness of the regressor. For each document, a vector of 4000 features is used. The details of the size of the training sets is given in the Table 5.1.

Crawling: During the training stage, the input data is split into training and validation data. Clustering based OR is then used to determine the regressor. The parameters η and W are tuned on the validation set by grid search. During the crawling phase, each newly crawled document is fed to the regressor which returned a label indicating the predicted level. Each document in Nalanda iVia Focused crawler implementation requires a relevance score, between 0 and 1, before it is processed by the crawler. The function used is a step function: relevance scores were 0.95,0.85,0.75,0.65, and 0.55 for level 0, 1, 2, 3, and 4 respectively. If a page is marked as level 0, 1, 2, or 3, the links from the pages were added to the priority queue; any page marked as level 4 is discarded. But only pages belonging to levels 0, 1 and 2 are considered as relevant pages when measuring the performance of the crawler. The score need not be based on only this function, it can be any kind of function whose range is $[0, 1]$.

Results: Comparison of the harvest rate graphs for categories NASCAR, Soccer, Cancer and Mutual Fund are show in figure 5.1, 5.2, 5.3 and 5.4 respectively. Table 5.2 gives a comparison of the performance of FOCUS with the baseline crawler. The number presented are the harvest rates as defined in Equation (2.1). As seen, FOCUS performs better than the baseline crawler on categories considered very difficult.

Table 5.3: Comparison of training times (in sec) between **SMO-OR** and **Clustering based OR** on benchmark datasets. (CH-California Housing, CS-Census datasets).

	S-Size	**SMO-OR**	**Clustering based OR**
		sec	sec
	10,320	551.9	112
CH	13,762	1033.2	768.8
	5,690	893	20.4
	11,393	5281.6	108.8
CS	15,191	9997.5	271.1
	22,331	$\times$	435.7

5.2 Clustering based OR

In this section we present results of scaling experiments on two large benchmark datasets and show that the clustering based OR algorithm (denoted by **CB-OR**) scales well to large datasets. The two benchmark datasets used are California Housing dataset[1] and Census dataset[2]. California Housing dataset has $20,640$ datapoints in 8 dimensions and Census dataset has $22,784$ datapoints in 16 dimensions. The training time and generalization of **CB-OR** is compared with the SMO algorithm[3] for OR formulation (4.3) [21] (denoted by **SMO-OR**). SeDuMi (denoted by **SeDuMi**) is used to solve clustering based OR formulation. The benchmark datasets were randomly partitioned into training and test datasets of different sizes to show the scalability of both algorithms. In all cases the results shown are for the parameters that gave best accuracy on the testset. Table 5.3 summarizes the scaling experiments on California housing dataset and Census dataset.

The results clearly show that the training time with **Clustering based OR** is very less when compared to **SMO-OR**.

[1] Available at http://lib.stat.cmu.edu/datasets/

[2] Available at http://www.liacc.up.pt/ ltorgo/Regression/DataSets.html

[3] http://www.gatsby.ucl.ac.uk/ chuwei/svor.htm

Chapter 6

FOCUS on top of Nutch

Nalanda iVia crawler is made to run on single computer and the implementation is a little bit completed. The focused crawler should run on distributed environment efficiently and the coding style should be simple to make it opensource. Nutch (http://lucene.apache.org/nutch/) is such a opensource crawler, but it is a general crawler. we implemented our focused crawling approach on top of the nutch code. Nutch uses MapReduce programming model [24], which is described in chapter 6.1. All the changes made by us follow MapReduce model, so that the code remains simple and scalable. Chapter 6.2 describes nutch architecture and focused crawling implementation is described in chapter 6.3

6.1 MapReduce

It is easy to parallelize the code automatically on large cluster of machines if the code is written in MapReduce model. The run-time system is responsible for partitioning the input data, scheduling programs on set of machines, handling failure of any machine and managing inter-machine communication. Programmer need not required to take care of all such details. Thus a programmer, who doesn't have much knowledge about parallel and distributed system, can also easily write a program, which utilizes resources of distributed system. Moreover whole dataset need not necessary to load into the main memory at a time, since the model processes each record independently. This property helps the model to work on real time large

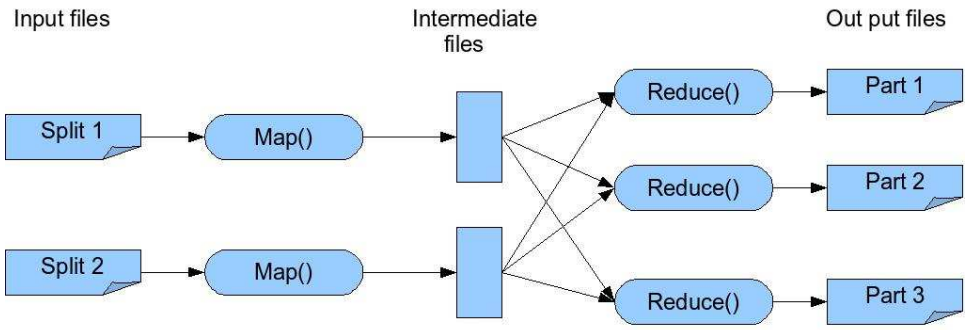

Figure 6.1: MapReduce model

dataset efficiently.

In MapReduce programming model programmer is required to provide only two functions: Map function and reduce function. As described in figure 6.1 first the run-time system partitions the input data into certain splits, one for each instance of map function. Several instances of the map function is allowed to run on different machines. The run-time system passes records from splits, one by one, to respective map function as key-value pairs. Thus programmer has to provide a map function which operates on individual records and outputs an intermediate key-value pair. Run-time system combines the values, which share same key. Now the key and vector of values are passed to a particular instance of reduce function. Thus programmer has to provide a reduce function which operates on single key and multiple values and outputs a single key-value pair. In MapReduce model map and reduce function operates on one key-value pair at a time. Many real-world tasks containing large data processing are easily expressible in MapReduce model.

Let us understand MapReduce model by an example. Let we have a large input file and we want to count how many times each word occurs in the file. The output file should contain $word - wordcount$ pairs. The run-time system splits the file and passes lines one by one to particular instance of map function. The map function takes a line as input and for each word in that line it generates intermediate key-value pair, where the key is respective word and the value is integer 1. The run-time system combines the values, which share same word(key). Thus if some word occurs three times in the input file then correspondind vector will be (1,1,1). For

each word run-time system passes the word and the corresponding vector to reduce function. Reduce function just adds the values in vactor and outputs a key-value pair, where value is the sum of the values in the vector. Run-time system writes such key-value pairs one by one to a output file. Thus programmer has to write just two simple functions, map and reduce, as described above.

The important point here is that we can run several map and reduce task parallely on several machines. If we want to use 3 machines for word count task, then run-time system makes instance of map and reduce function on each machine. The input file is partitioned into 3 splits. Each map task is responsible for one of the 3 splits. When all map function complete their task then run-time system combine the values which share the same word. Rules for partitioning the intermediate $key - (valuevector)$ pairs can be defined like words starting from 'a' to 'i' go to first reduce task, words starting from 'j' to 'o' go to second reduce task and remaining words go to third reduce task. When all reduce tasks complete their work, the run-time system combines all output files.

Thus If we have several machines then MapReduce model can easily speed up the process. Moreover programmer has to write just two easy functions: map and reduce. Run-time system takes care of distributed environment. Programs written using MapReduce model scale up well because only one key-value pair is required to stay in the memory at a time.

6.2 How nutch works

Nutch is an opensource software, which contains modules for crawling, indexing and searching. User is required to provide a file containing seed urls. Nutch crawls predefined number of web pages starting from the given seed urls.

As shown in figure 6.2 first injector injects seed urls into crawl database as unfetched urls. Crawl database contains url-crawldatum as key-value pair, where crawldatum contains necessary information about url: whether it is fetched, unfetched or linked, last fetch time, signature, etc. Now generate-fetch-parse-update cycle runs $depth$ times, where $depth$ is a user defined parameter. In each cycle a new segment is generated, which contains all the required

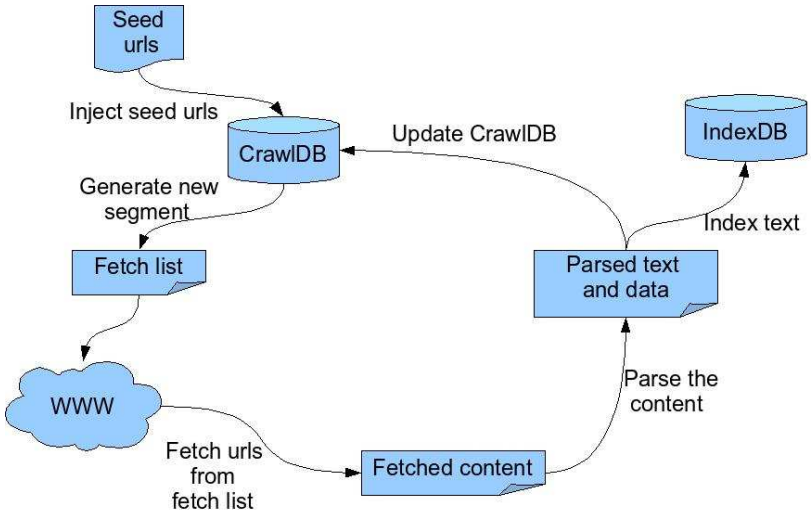

Figure 6.2: Nutch Architecture

data related to topN best score urls at the segment generation time.

Generator module selects $topN$ unfetched urls from crawl database and puts in fetch list for newly generated segment, where $topN$ is user defined parameter. As described in algorithm 1 the generator task is divided into two MapReduce tasks. The first MapReduce task selects $topN$ best score urls. It processes url-crawldatum records from crawl database. If status of the url is $notfetched$ then only the map function passes the record as $(inlinkscore) - (url, crawldatum)$ pair to the run-time system. Run-time system sorts all the records based on inlink score and passes them to reduce task in decreasing order. Reduce task outputs only first topN records. Seconds MapReduce task is required to prepare the records as $url-crawldatum$ pairs instead of $(inlinkscore) - (url, crawldatum)$ pairs.

Fetcher module fetches all the urls from the fetch list and store the web pages in content database. Parser module parses (algorithm 2) content of web pages and generates three separate databases: One database contains $url - parsedtext$ pair, which is required for indexing, second database contains $url - parseddata$, which stores $link - outlink - anchortext$ details and is required for update module and third database contains $url - crawldatum$ pair for fetched urls in previous fetch task. Information from previous crawl database, parsed data (second database from parse module) and crawldatum data (third database from parse module) are combined into one new crawl database by update module.

Algorithm 1 GENERATE

MapReduce part1 : Select urls due for fetch

Input: CrawlDb file

Map()-If $date \leq now$, collect (CrawlDatum, url)

Partition by hash value

Reduce() - Order by decreasing link count

- Output only topN entries

MapReduce part2 : Prepare urls for fetch

Map() - Invert key-value pair

Partition by host

Reduce() - Identity function

Output: Set of (url, CrawlDatum) files

Algorithm 2 PARSE

Require: (url, Content) file from fetch

Map() - Parse the content using parser plugin

- Collect (url, Parse)

Reduce() - Identity

Output: 1) (url,ParseData), 2) (url,ParseText), 3) (url,CrawlDatum)

Once the crawling is completed, invertlink module inverts all links using parsed data to get anchor text, which is required for indexing. Indexer module generates index using anchor text generated by inlink module and parsed text generated by parser module.

6.3 Focused Crawling Implementation

Nutch software is a general crawler. To make it focused crawler we plugged our ordinal regression module in the nutch code and changed generator module such that it generates topN most relevant urls in the fetch list.

We assume that user will provide level files along with the seed file. We train ordinal regression before starting focused crawling. As described in algorithm 3 training module processes level files one by one. Injector injects urls from level file into the crawl database of respective level. Generator creates a new segment and generates fetch list which contains all urls from respective crawl database. Parser module parses web pages and stores parse text in one file.

We used MapReduce model to convert text into raw vector file as described in algorithm 4. Map function is identity function. Reduce function receives parse text of one web page at a time. It converts text into raw vector format and stores new words in word list, which also contains document count (Number of documents containing that word) . Raw vector format is required for CBOR algorithm.

Once all levels are processed, all raw vector files are combined into one file. Now the raw vector file is converted into tf-idf vector file using word list. Chi-squared feature selection method is used to reduce the number of feature. Ordinal regression is trained using only selected features.

Parse module is changed as shown in algorithm 5 to assign orscore of parent page to all out links. After parsing the web page, test method of ordinal regression module is called to get orscore for that page. The orscore is assigned to all the outlinks of that page, which are then stored in parse data database. Generator module is changed to make sure that it generates topN urls based on orscore instead of inlink score.

Algorithm 3 TRAINING

Require: level files

for all levels **do**

Run INJECT, GENERATE, FETCH and PARSE module

Run FEATUREGENERATE module

end for

Combine all raw vector files

Dump word list

Run CBOR module

Algorithm 4 FEATUREGENERATE

Require: parse text file

Map() - Identity

Reduce() - Convert text into raw vector formate

Output: Raw vector file

Algorithm 5 NEWPARSE

Require: (url, Content) file from fetch

Map() - Parse the content using parser plugin

- Get orscore by testing page text

- Set orscore in all the outlinks(ParseData)

- Collect (url, Parse)

Reduce() - Identity

Output: 1) (url,ParseData), 2) (url,ParseText), 3) (url,CrawlDatum)

Our new module and all the changes follow MapReduce model. It is necessary because if one module does not follow MapReduce model then that module will slow down the whole system and new system will not be scalable any more.

Chapter 7

Conclusions

A focused crawler that makes use of the linkage structure of the web similar to [12] is presented in the thesis. It uses openly available general search engine APIs provided by Google to back-crawl and construct a local neighborhood of the seed pages. This local neighborhood is used to learn the web structure, that when traversed leads to topic pages. An ordinal regression formulation is proposed to rank the URLs that need to be crawled by the crawler. A large scale clustering based ordinal regression formulation is proposed to handle large seed sets.

A working system was setup on top of Nalanda iVia Focused crawler. Experiments done on the base system and the modified system with the ordinal regression formulation justify the ideas presented. Independence of crawler strategy from document taxonomy and notion of negative class provides distinct advantages to the method. It only needs the existence of a mechanism to get the back-link structures for smooth running of the focused crawler. This approach to focused crawling can help in creating focused crawlers using seed pages derived from a set of queries. This method provides a direction for design of focused crawlers which can crawl pages related to any event rather than a topic.

The current system is implemented on top of Nutch opensource crawler, which provides search engine also. Nutch code is made using MapReduce programming model. All the newly added modules and changes to old modules, to make it focused crawler, follows MapReduce programming model. The new focused crawler implementation can be run on cluster of machines and is scalable. It can be used for large scale web crawling also.

Future work: The current system assigns equal priority/relevance score to each URL in a given document. A strategy that uses the documents contents and the URL tokens to assign different priority/relevance scores has already been studied in [10, 11]. Moreover the crawler does not use the DOM structure to prioritize the URLs in the document as in [10, 11]. Usage of such strategy would increase the accuracy of the simple model of the web being used in this project report. The ordinal nature of the formulation provides many opportunities of generating relevance scores to the crawled pages. Different scoring mechanisms need to be looked into and evaluated. A Reinforcement Learning based feedback mechanism that continuously learns the structure of the web for the topic needs to be incorporated which will further improve the crawler.

Bibliography

[1] J. Saketha Nath, C. Bhattacharyya, and M.N. Murty. Clustering based large margin classification: a scalable approach using SOCP formulation. *Proc. of 12th Intl. Conf. on Knowledge discovery and data mining*, pages 674–679, 2006.

[2] Ivia. The Nalanda iVia Focused Crawler. *Code at http://ivia.ucr.edu/*, 2006.

[3] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1998.

[4] J.M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[5] B. Novak. A Survey of Focused Web Crawling Algorithms. *SIKDD 2004 multiconference IS 2004*, pages 12–15, 2004.

[6] P. De Bra, G.J. Houben, Y. Kornatzky, and R. Post. Information Retrieval in Distributed Hypertexts. *Computer Networks and ISDN Systems*, 27(2):183–192, 1994.

[7] M. Hersovici, M. Jacovi, Y.S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. The Shark-Search Algorithm. An Application: Tailored Web Site Mapping. *WWW7 / Computer Networks*, 30(1-7):317–326, 1998.

[8] J. Cho, H. Garcia-Molina, and L. Page. Efficient Crawling Through URL Ordering. *WWW7 / Computer Networks*, 30(1-7):161–172, 1998.

[9] S. Chakrabarti, M. van den Berg, and B. Dom. Focused Crawling: A New Approach for Topic-Specific Resource Discovery. *WWW Conference*, 1999.

[10] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. *Proc. of 11th Intl. Conf. on World Wide Web*, pages 148–159, 2002.

[11] C.C. Aggarwal, F. Al-Garawi, and P.S. Yu. Intelligent crawling on the World Wide Web with arbitrary predicates. *Proc. of 10th Intl. Conf. on WWW*, 2001.

[12] M. Diligenti, F. Coetzee, S. Lawrence, C.L. Giles, and M. Gori. Focused crawling using context graphs. *Proc. of 26th Intl. Conf. on VLDB*, 2000.

[13] H. Liu, E. Milios, and J. Janssen. Focused Crawling by Learning HMM from User's Topic-specific Browsing. *Proc. of the Web Intelligence*, pages 732–735, 2004.

[14] D. Grangier and S. Bengio. Exploiting Hyperlinks to Learn a Retrieval Model. *Proc. of NIPS Workshop*, 2005.

[15] B.D. Davison. Topical locality in the Web. *Proc. of 23rd Intl. Conf. on Research and development in Information Retrieval*, pages 272–279, 2000.

[16] Google. Google SOAP Search API^{TM}. *Code at http://code.google.com/apis/soapsearch/*, 2006.

[17] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers*, pages 115–132, 2000.

[18] K. Crammer and Y. Singer. Pranking with ranking. *NIPS*, 14, 2002.

[19] A. Shashua and A. Levin. Ranking with large margin principle: Two approaches. *NIPS*, 15, 2003.

[20] W. Chu and Z. Ghahramani. Gaussian Processes for Ordinal Regression. *The Journal of Machine Learning Research*, 6:1019–1041, 2005.

[21] W. Chu and S.S. Keerthi. New approaches to support vector ordinal regression. *Proc. of 22nd Intl. Conf. on Machine learning*, pages 145–152, 2005.

[22] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. *Proc. of Intl. Conf. on Management of data*, pages 103–114, 1996.

[23] J.F. Sturm. Using sedumi 1.0 x, a matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11:625–653, 1999.

[24] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplied data processing on large clusters. *OSDI 04*.