Indian Institute of Technology Bombay

B.Tech Project Stage I

Web Crawling and IR

Author:
Naga Varun Dasari

Supervisor:
Dr. Pushpak Bhattacharya

A thesis submitted in partial fulfilment of the requirements for B.Tech Project Stage I

in the

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

November 2013

Acknowledgements

I would like to take this opportunity to express my gratitude towards my guide Prof. Pushpak Bhattacharyya for his motivating support, guidance and valuable suggestions. I would also like to thank Arjun Atreya for sharing his experience in this subject. I would also like to thank Swapnil, Jayaprakash, Biplab, Sanket, Rohit and other CLIA team members who have listened to my weekly presentations and gave constructive criticism and feedback.

Abstract

Web Crawling is the first and foremost stage in any web Information Retrieval system. Though the basic architecture appears simple, there are many optimizations that should be done to hardware resources and decisions to be taken related to various algorithms and policies that should be implemented, for efficient and cost-effective crawling. The aim of the project is to learn about web crawling and some of its variants and some basic concepts of ranking to lay the foundation for future work in the second stage. In this work, the major part constitutes literature survey on these topics. Specifically, we discuss crawler architecture and one of its most important variation - Focused Crawling and its application in language specific crawling. We also discuss how relevance of search results can be improved by using personalized Information Retrieval. Before going into that we will discuss the Probabilistic Ranking Principle and its shortcomings which led to new ranking principles. In the end, we discuss some experiments related to language specific crawling done on Nutch.

Contents

List of Figures

Chapter 1

Introduction

1.1 Crawling

The content in the web is not managed by a single person but consists of millions of people managing their respective content. During the beginnings of the Internet, there were two models for the content aggregation. One was the Pull model, where the search engines and other content aggregators employ a process which actively checks for new and updated pages. The other was the Push model where they could enable the content providers to push content to them. But the Push model eventually faded out and Pull model became almost the only model of web crawling. Push model did not succeed because it raised the barrier of entry into web as the content providers had to follow protocols for pushing information to the aggregators and some trust had to be established between the provider and aggregator before an aggregator accepts to include its information in the search index. Whereas in pull model, the provider just needs to put his content in the server and need not care much about whether the search engine is showing his website in the search results,unless he wants to prevent some crawlers from accessing certain parts of the website. For this purpose a protocol called robots exclusion protocol has been developed and the crawlers are expected to follow the guidelines provided in it. The Pull model has gradually evolved into what we now call as Web crawling. The basic architecture of web crawling appears very simple, but there are many optimizations that should be done to the algorithms, data structures and also the hardware that are used. We are going to discuss in detail about the architecture of web crawler in further chapters.

1.2 Focused Crawling

Focused Crawler is a variation of a basic crawler which selectively collects the web pages satisfying certain properties. For example, if we need to crawl web pages only from '.in' domain or only of a particular language like Hindi or pertaining to a specific topic like Tourism, we need to employ a focused crawler. Topic specific crawler is the most important and highly studied variation of Focused Crawler and all other types of Focused Crawler mostly employ the methods of Topical Crawling. In the early days of web crawling, the advancements in computational power were in budding stage. This posed a limit on the scale of a generic crawler. It used to take weeks to a month to update a crawl, and were able to cover 30% to 40% of the whole web. In spite of this effort, it becomes difficult to give relevant results unless the query is constructed correctly. This led to the idea of constructing a Focused Crawler and using many of them in different computers each responsible for a certain topic. By giving a semi structured query which contains the information about the topic in which the user intends to see the results, we can perform the search only in the index of that topic resulting in an increase in relevance of results, at the same time reducing the time to perform the crawl using the same technology as before. We are going to discuss in detail about Focused Crawling in Chapter 3.

1.3 Probabilistic Ranking Methods

As the name implies, the probabilistic ranking methods employ probability theory for ranking the results in an IR system. These are some retrieval methods which do not use probability theory or ranking in general:

- **Boolean Retrieval Model:** Query is in the form of a boolean expression. Ex: Query 'Ramu AND Shamu' means both 'Ramu' and 'Shamu' must be present in the retrieved documents. Document is in the form of Term-document matrix A where $A_{ij} = 1$ if and only if term t_i is present in j^{th} document. The documents are not ranked according to the relevance.
- **Vector space model:** Query and documents are vectors in space of terms. The cosine similarity between document and query is considered as the relevance score for ranking the documents. Document vectors are calculated using TF-IDF model.

The information needs of a user are represented by the query which is used for retrieving relevant documents. As there is an inherent uncertainty regarding whether a document

is relevant with respect to a given query, we can say that probability theory acts a fundamental tool to measure the uncertainty. We are going to discuss Probabilistic Ranking Principle and some models and some variations of the principle in Chapter 4.

1.4 Aim and Motivation

Web search engines are based upon the huge corpus built by storing maximum possible web pages relevant to the domain for which it is intended to retrieve results. These pages are collected by a web crawler and the collected web pages are analyzed to strip down the irrelevant parts (e.g., html tags, ads etc.) and the relevant portions are indexed. When a user submits a query to a search engine, it uses its index, and returns the best web pages relevant to the query. In order to provide the most relevant documents, search engines employ various methods to rank the results.

Many small scale and academic organizations have resource constraints to build a full blown web search engine. Crawling and ranking techniques which are developed for large scale information retrieval systems may become difficult to implement in small scale search engines. Learning about current crawling and ranking principles builds the foundation for further research on possible modifications that can be made for small scale search engines. In this report, we discuss about crawling and some probabilistic ranking models. We deeply go into focused crawling which is used for crawling only the pages relevant to particular domain.

1.5 Overview of the Report

In chapter 2, we discuss the basic architecture of web crawling. We list out some desired features in crawling and describe the architecture and how it is able to satisfy the features. In chapter 3, we discuss some methods of how a focused crawler can be implemented. As web page classification comes into play in focused crawling, we describe some techniques to perform the classification. Then we go on to describe language specific crawling, which is used to collect web pages of a particular language. In chapter 4, we go into ranking aspect of IR by discussing Probabilistic Ranking Principle and some of its variants. In chapter 5, we dwell upon the topic of personalized IR and how it improves the relevance of result set by taking the previous history of the user into account. In chapter 6, we discuss various evaluation methodologies which are prominently used to test the effectiveness of an IR system. In chapter 7, an implementation of Telugu language specific crawling and some observations are discussed. In the end, we present some conclusions and directions for future work.

Chapter 2

Crawling

In this chapter, we are going to discuss the architecture of crawling. In the first section, we will list out some challenges and desired features in a web crawler. In the next section, we will discuss the architecture which naturally fulfills the features given in the first section. We will then see how it can be extended to a distributed crawler.

2.1 Challenges and Desired Features

The basic algorithm of crawling can be described as follows: Given a set of initial URLs(Uniform Resource Locators), the crawler downloads all the corresponding web pages and extracts the outgoing hyper-links from those pages and again recursively downloads those pages. Some challenges involved in Crawling are:

- **Scale:** Crawlers which are supporting search engines for the entire web must have extremely high throughput in order to maintain freshness and cover maximum possible content. The search engines like Google, Bing etc. use large number of computers and high bandwidth networks for this purpose.
- **Content selection trade-offs:** Crawler should not waste time on low quality and irrelevant pages. It should ensure coverage of quality content and maintain a balance between coverage and freshness.
- **Follow rules:** The crawler should not repeatedly download a web page in order to maintain freshness. Otherwise it unintentionally causes a denial of service attack. A time interval must be maintained between consecutive requests to a web page depending on the capacity of the host the web page is residing in. Some websites

do not want the crawler to download a particular portions of the website. Crawlers should obey such rules imposed by a website.

- **Dodge misleads:** Some web pages have near duplicate content or just mirrors of some well known pages. Some web pages appear to be constantly changing but mostly this is due to the ads or comments by readers. The crawler should be able to detect such anomalies.

2.2 Crawler Architecture

The basic architecture of a web crawler is shown in Figure 2.1. The important modules involved in a crawler are:

- **URL Frontier** contains the list of URLs that are yet to be fetched.
- **Fetch module** actually fetches the web page.
- **DNS resolution module** determines the address of the server from which a web page has to be fetched.
- **Parsing module** extracts text and links from the fetched web page.
- **Duplicate elimination module** eliminates the URLs which are already present in URL Frontier.

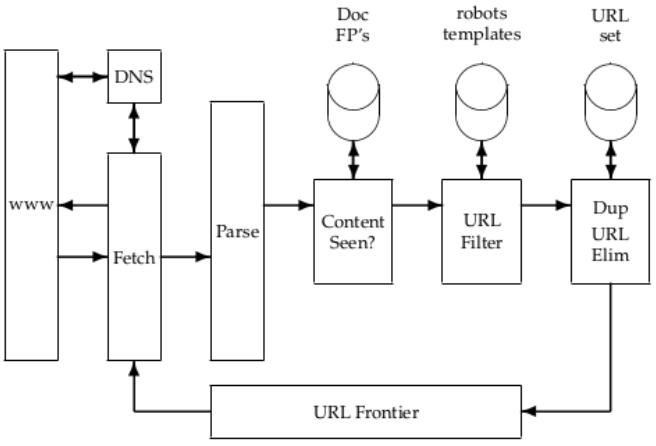

FIGURE 2.1: Basic architecture of crawler

The crawling process consists of several threads and each thread performs work cycles repeatedly. The following is the brief overview of a work cycle: At the beginning of the work cycle, the worker thread obtains a URL from URL Frontier. The Frontier pops

out a URL according to its priorities and politeness policies. Then the Fetcher module is called which calls DNS module to resolve the host address of the corresponding web server. After obtaining the address, the Fetcher module connects to the server and checks for robots exclusion protocol(described later) and tries to download the web page. If the download is succeeded, the web page is sent to Link Extractor which extracts the outgoing links. The extracted URLs are passed to the URL filter which filters out some unwanted URLs such as file extensions of no interest, black listed sites etc. These filtered URLs are passed to Duplicate eliminator which removes the URLs already present in the Frontier. Finally the URLs reach the URL Frontier which allots them specific positions in its data structure according to some fixed priority rules.

In this section, we do not discuss the Fetcher and Parser modules in detail, as they are fairly straightforward except for some decisions to be taken regarding politeness policies in the Fetcher module. We will discuss in detail the implementation of URL Frontier, DNS resolution module and Duplicate Elimination module.

2.2.1 URL Frontier

The web crawler has to maintain a data structure that keeps track of the URLs that are to be downloaded. This is the main part of URL Frontier. This data structure must support the following operations: Adding a URL and selecting a URL to fetch.

The basic implementation consists of a First-In-First-Out queue which results in breadth first traversal of the web. But there are disadvantages to this idea: Since many web pages contain links referring to the web pages in the same site, the FIFO queue contains long runs of URLs pointing to the same web site which results in an unintentional denial of service attack on the web server. Also the crawler would be spending too much time on a single website without exploring others.

Now we describe an implementation which prioritizes the pages and follows politeness policies in giving the page to be fetched next. Priority must be given to high quality pages with high rate of change. Politeness implies avoiding repeated requests to a host within a short span of time. Figure 2.2 shows the outline of such implementation. It is divided into a front end and back end. Front end consists of F FIFO queues and back end consists of B FIFO queues. When the URL is added to the Frontier, the prioritizer assigns a priority from 1 to F to the URL depending on rate of change in the previous crawls and some other factors. URL which is assigned priority i is pushed into i^{th} queue.Each queue in back end consists of URLs belonging to a single host. A table is maintained that tells which queue holds the URL of a particular host. A time t is associated with each queue at which the next URL from the host can be requested.

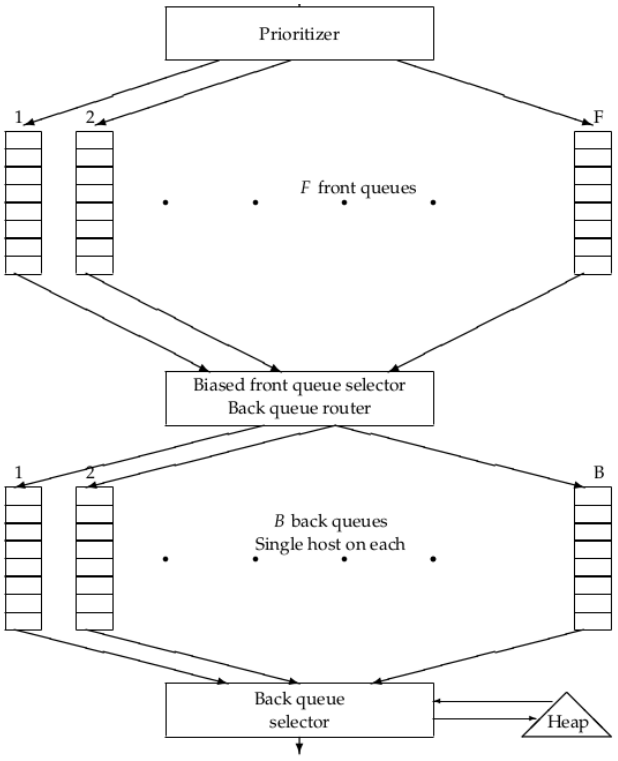

FIGURE 2.2: URL Frontier

These times are maintained in a priority queue. This time depends on the fetch time for the last URL fetched from that host and the time when it was requested. When the worker thread requests the next URL from the frontier, the frontier takes the head of the priority queue and outputs the URL from the head of corresponding back queue. If the back queue becomes empty, the following process is repeated until the queue becomes non-empty: a front queue is selected, by a random process biased towards higher priority queues, and the URL at the head of the queue is selected and if there exists a mapping in the table corresponding to the host of that URL, it is pushed into the corresponding back queue. If there is no entry in the table, the host is assigned to the empty back queue and the URL is pushed into the back queue and the process is terminated.

In-memory crawler does not scale efficiently. So we need to resort to disk based implementations and reduce the number of disk seeks as disk seeks take order of magnitude larger time than in-memory access. One thing we could have done in the URL Frontier is to enable to assign any arbitrary priority to each URL in the front end and maintain a single priority queue, instead of having some fixed priority levels. But this becomes inefficient in disk based implementations because each dequeue potentially requires O(log n) disk seeks which is bad if we want high speed web crawling. This is why we fixed the number of priority levels so that only a part of front of each queue needs to be in memory any time.

2.2.2 Duplicate URL elimination

The web crawler has to maintain another data structure to store the URLs which are currently in the frontier. This data structure has to support the following operations: adding a URL, checking whether a URL is already present in it and deleting a URL. One obvious implementation is a hash table. Since memory would not be sufficient on large scale we need to employ disk based implementations but disk based hash tables take much longer than in-memory ones. One solution is to cache the frequently seen URLs which helps in reducing the number of accesses to the hash table in disk. Another solution is to aggregate the lookup and insertion operations ordered by their position on disk, so that it essentially becomes a one long sequential read and write operation which takes considerably lower time than performing random disk seeks. But there is one small disadvantage to this. We cannot immediately add a URL to the Frontier, we have to wait for the batch operations to complete. However the Frontier is usually large enough that there are sufficient number of URLs to keep the crawling going before adding the new ones. But the most high priority and highly changing web pages are affected.

2.2.3 DNS Resolution

The address of the host component of a URL is found using Domain Name Service(DNS). The protocol is recursive in nature which can potentially take seconds in some cases, which immediately hampers our goal of fetching several documents a second. A common solution for this problem is to use the DNS cache of the machine in which we are performing the crawl. But the general standard implementations of DNS lookup are synchronous in nature, which means all the crawl threads come to a halt until the requested address is obtained. In order to avoid this problem, the crawlers implement their own DNS resolver threads. The worker thread sends the request to DNS server and performs a timed wait: resumes when signaled by another thread or when a fixed time quantum expires. Another thread listens on the standard DNS port for a response and as soon as it receives a response, sends it to the corresponding thread and signals it. If the time quantum completes, the crawler thread sends another request to the DNS server and performs timed wait with an increased time quantum.

2.2.4 Robots Exclusion Protocol

A protocol has been developed to communicate information from the host to crawlers about its policy of not allowing certain or all crawlers to crawl certain web pages within

the site or the whole web site. This protocol is called Robots Exclusion Protocol and HOST/robots.txt is the place where this information is provided. Downloading this page is also essentially crawling itself. But since this information is required repeatedly for every URL from that host, it makes sense to maintain a cache for these files. We have to decide on cache eviction policy(like Least Recently Used policy) and some web servers themselves provide an expiration date for their robots.txt file.

2.3 Distributed Crawling

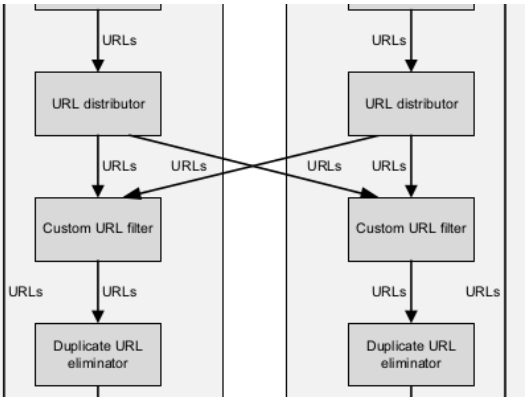

FIGURE 2.3: URL splitting in Distributed Crawler

In distributed crawling, each or some worker threads are made to run on a different machine or node so that the throughput is increased. Each node is responsible for certain hosts. Partitioning the hosts among various hosts can be based on various policies. One policy is based on geographical location. Because a web page generally contains majority of links to a page of same site, most of the extracted URLs fall under the responsibility of the same node. The URLs whose responsibility falls under other nodes have to be forwarded to the respective nodes. Specifically speaking we can have a URL splitter before the Duplicate Elimination module which takes care of this. There exist various replicas of some famous web pages in multiple web sites. In order to determine whether a web page is a copy of a previously seen web page, we should maintain a finger print or shingles of each web page. Distributed crawling becomes tricky if we want to detect such replicated web pages. For checking whether a page is a replica, it is not sufficient to check for replica from the pages crawled by the same node. It might be the replica of a web page crawled by some other node. So we have to check for the web pages from all the nodes. One possible idea is we can do batch lookup which reduces the number of network requests.

2.4 Summary

In this chapter, we have discussed the architecture of crawling. We have explained in detail how URL Frontier, DNS resolution module, Duplicate URL elimination module are implemented. Then we have seen how a normal crawler can be extended to a Distributed Crawler.

Chapter 3

Focused Crawling

In this chapter, we are going to discuss about focused crawling. In the first section we are going to discuss the Focused Crawling mechanism given by Chakrabarti et al.[1]. In the second section, we will see some shortcomings of this mechanism and discuss an alternative method proposed by Diligenti et al.[2]. Since webpage classification comes into picture in focused crawling, in the next section we will describe hierarchical classification technique proposed by Dumais et al.[3] and positive example based learning technique given by Chen et al.[7]. Then we go on to discuss an application of focused crawling, language specific crawling. We describe a variant of Focused Crawler proposed by Somboonviwat et al.[5] and then go into a method to detect Indian languages in web pages proposed by Pingali et al.[6]

3.1 Using Hierachical Structure of the Web

The term Focused Crawler was coined by Chakrabarti et al. [1] and the crawler mechanism proposed by them used a classifier in the crawl frontier. The term Focused Crawler has actually became a synonym of Topical Crawler and is more widely used. The focused crawler is based on a classifier which recognizes the relevant pages from the examples in a hierarchical topic taxonomy.

3.1.1 User's View

These are the various steps involved in the Focused Crawling from the perspective of user:

- **Canonical Taxonomy Creation:** The initial system consists of a classifier which is trained with a canonical taxonomy such as the one in Open Directory Project and a set of examples corresponding to each topic.
- **Example Collection:** The user selects URLs of his/her own interest.
- **Taxonomy selection and refinement:** The URLs provided by the user are classified with the initial classifier and the results are shown to the user. The user marks the classification results which feel correct to him as 'good' and correct the wrongly classified ones and mark them as 'good'. The crawler should focus on collecting pages similar to the documents marked as 'good'.
- **Interactive exploration:** The system then proposes some URLs which are closer to the examples provided by the user for classification. The user may classify some of them manually and mark some of them as 'good' which means they are relevant.
- **Training:** The classifier incorporates the propositions made by the user to train itself.
- **Resource discovery:** The crawling process starts.
- **Distillation:** In between, concurrently, a distillation algorithm is run to identify hubs. Hubs are the pages which contain large number of links to the relevant pages.
- **Feedback:** The hubs list is shown to the user intermittently which are marked as relevant or non-relevant and this feedback goes back to the classifier.

3.1.2 Classification

The probability that a web document d belongs to a topic c is estimated using a hypertext classifier. Since the taxonomy is hierarchical, if we consider each topic c as a node, each node has a parent denoted by *parent(c)* except the root. The probability is estimated as follows:

$$P(c|d) = P(parent(c)|d)P(c|d, parent(c)) \tag{3.1}$$

The first term of the RHS in the above equation is calculated recursively. The second term can be calculated using Bayes rule:

$$P(c|d, parent(c)) = \frac{P(c|parent(c))P(d|c)}{\sum_{c' \in siblings(c)} P(d|c')} \tag{3.2}$$

$P(c|parent(c))$ is the prior distribution of documents. For $P(d|c)$, a document generation model is needed. A document d is assumed to be a bag of words and if a term t occurs $n(d,t)$ times, then:

$$P(d|c) = \binom{n(d)}{n(d,t)} \prod_{t \in d} \theta(c,t)^{n(d,t)} \tag{3.3}$$

Depending on whether we completely ignore the irrelevant page or not, the following are the two modes of focused crawling:

- **Hard Focus:** The above formulae are used to find the topic c* with the highest probability that the document belongs to that topic. If an ancestor of c* is marked good, then the extracted URLs are added to the frontier, otherwise the crawl is pruned at d.
- **Soft Focus:** The probability that a page is relevant to our goal is calculated using:$R(d) = \sum_{good(c)} P(c|d)$. Then the priority of the extracted URLs from this page is assigned according to this relevance.

3.2 Using Context Graphs

The major problem in the method given in the previous section is that if a page crawled in the current level is irrelevant but there are some pages which are relevant at a certain distance from it, the chances of covering those pages will become significantly low. Diligenti et al.[2] proposed an algorithm which models the context around the topically relevant pages in the web. This algorithm uses the partial reverse crawling capabilities of search engines.

3.2.1 Generating Context Graphs

A separate context graph is built for each seed URL. The seed starts as the root. Using a search engine, the pages linking to the seed are retrieved which can be called parent pages. These are added into the graph with an edge to the root. The reverse crawling and forming layers is repeated for fixed number of levels N. After few levels, the number of pages are exponentially increased in which case, a sample of pages is retained. The context graphs of all the seed URLs are combined to form a Merged Context Graph.

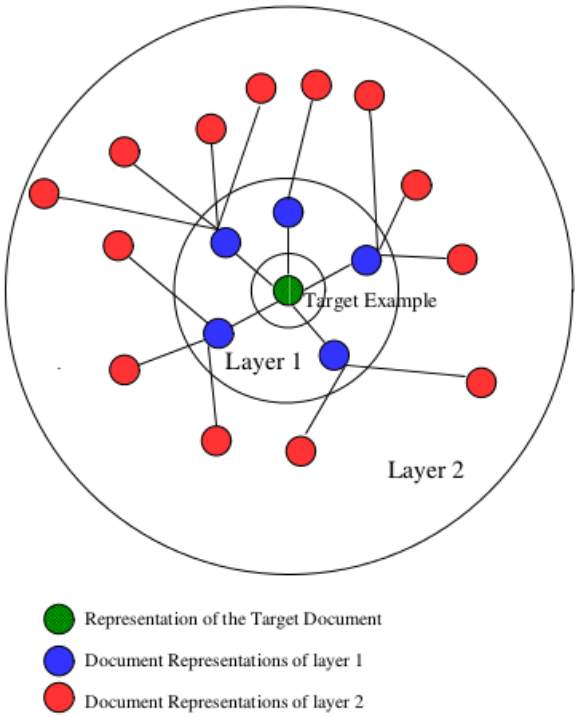

FIGURE 3.1: Context Graph

3.2.2 Classification

Each document is represented as a feature vector with features as the TF-IDF frequency of the words from a fixed vocabulary. Then each document is classified to be belonging to one of the layers or none. The vocabulary for each layer is built as follows: All the documents of the seed set and optionally first layer are combined, all the stop words are removed, word stemming is performed, TF-IDF is performed with respect to the corpus obtained from extensive web crawling. The words with higher TF-IDF frequencies are included in the vocabulary. For each document, TF-IDF frequency for the words in the vocabulary is calculated and the words with top 40 scores are retained and remaining are made zero. A modification of Naive Bayes Classifier is used for each layer. We find the probability that a document d_i belongs to class c_j using Bayes rule and employing Naive Bayes assumption:

$$P(c_j|d_i) \propto P(c_j)P(d_i|c_j) \propto P(c_j)\prod_{k=1}^{N_{d_i}} P(w_{i,k}|c_j) \tag{3.4}$$

where $w_{i,k}$ is the k$^{\text{th}}$ term in the vector representation of the document d_i

$P(w_t|c_j)$ is calculated by normalizing the number of occurrences of w_t in class c_j with total number of words in class c_j.

A confidence threshold is set for each layer so that if a document's probability of belonging to all the classes does not exceed the corresponding thresholds, it is classified as belonging to none of the classes.

3.2.3 Crawling

The downloaded URLs are maintained in N+2 queues. Queue 0 stores all the retrieved topically relevant documents. Queue N+1 has documents labeled as 'other'. Initially, all the starting URLs are placed in the queue N+1. All of them are downloaded and classified and placed in the queues corresponding to the winning layers. Subsequently, the following process is repeated: The crawling thread extracts the URL from the first non-empty queue and then downloads the pages pointed to by the URLs in that page and classifies them.

3.3 Web Page Classification Techniques

Web page classification is important to many information retrieval tasks. This is different from traditional text classification due to the humongous nature of web and interconnected nature of pages via hyperlinks.

3.3.1 Applications

- **Development of web directories:** The web directories provided by Yahoo! and dmoz (Open Directory project) make it possible to easily obtain example web pages belonging to a certain category. Web page classification helps to automate the process of classifying the pages instead of doing it manually.
- **Improving quality of search results:** One of the most important factors that affect relevancy of search results is the query ambiguity. One method is to ask the user at query time to specify one or more categories in which user is expecting the results. In situations where the user does not know the categories beforehand, one option is to classify the search results and present the categorized view to the user.
- **Focused Crawling:** In the previous section we have seen some methods of focused crawling in which a classifier is used to evaluate the relevance of document to given topics.

3.3.2 Hierarchical Classification

In non-hierarchical classification, generally, a separate binary classifier is used to distinguish one class from all other classes. The training of a classifier must be done against all other classes. Such training becomes difficult when there are large number of classes and features. Dumais and Chen [3] proposed that by utilizing a known hierarchical structure, the training can be decomposed into smaller sub-problems. Initially training is performed on the first level classes and the lower level training is done only among its siblings. This method uses Support Vector Machine(SVM) (binary feature version) which was used first time for hierarchical classification. The main aim of the authors was to automatically classify web search results during run time. So short summaries returned by search engines was used for classification and not the whole content of the web page. Summary consists of the title, the keywords, and either the description tag if it existed or the first 40 words of the body. The experiments are performed on top levels of hierarchy: 13 first level and 150 second level categories. The text was translated to lower case, words were separated using white space and punctuation as demarcators which are filtered using a stop word list. The features in the document vector representation are: 1000 words with highest mutual information with each category. The Mutual Information between a feature F and class C is defined as

$$MI(F,C) = \sum_{F\in\{f,\overline{f}\}} \sum_{C\in\{c,\overline{c}\}} P(F,C) log \frac{P(F,C)}{P(F)P(C)} \tag{3.5}$$

During the training stage, a large set of web pages with known category labels are used to train a classifier. During the classification process, compute the probability of the document being in each of the 13 top-level categories and each of the 150 second-level categories. There are two ways to combine probabilities from the first and second level for the hierarchical approach. In one case we first set a threshold at the top level and only match second-level categories that pass this test. In other case we compute product of probabilities. The accuracy was measured using F1 value = 2*P*R/(P+R).

3.3.3 Positive Example Based Learning

Constructing binary classifier requires both positive and negative training examples. But the collection of negative examples is a tough process. Negative examples must uniformly represent the universal set excluding the instances belonging to positive classes. Manual collection of negative examples may contain bias toward a particular type of negative examples. Yu et al. [4] proposed PEBL framework which just needs positive and unlabeled data. The algorithm proposed is called Mapping-Convergence (M.C)

algorithm which achieves classification accuracy of traditional SVM with positive and negative data. Before describing the M-C algorithm, we will define the following notations: POS : positive data set, $M_i(neg)$: A subset of negative examples where $M_i(neg)$ is farther to POS than $M_{i+1}(neg)$. $M_1(neg)$ is the subset of negative examples farthest from POS. $M_n(neg)$ is the closest to POS.

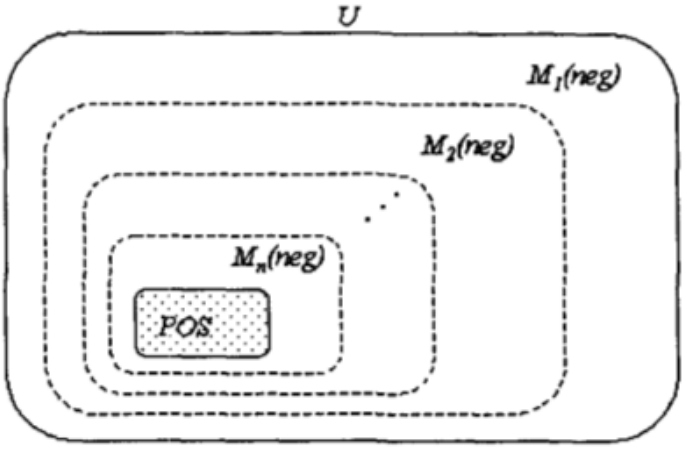

FIGURE 3.2: Visualisation of the sets

- **Mapping stage:** In this stage, the strongest negative data is found using 1-DNF learning. This means we will list all the features that appear in most of the positive data but occurs rarely in unlabeled data. Then we filter out every possible positive data point from unlabeled data which leaves us with only strongly negative data which is M1(neg). This is our current NEG.
- **Convergence Stage:** An SVM is built with POS as positive data and NEG as negative data. This SVM is tested on S1(pos) which is divided into M2(neg) and S2(pos). Now the NEG incorporates M2(neg) and the SVM is retrained using the new POS and NEG. The process is repeated iteratively until Mi(neg) becomes empty.

3.4 Language Specific Crawling

In language specific crawling, our aim is to crawl only those web pages belonging to a particular language. There are two important issues involved in language specific crawling: One is automatic identification of language and the other is to have a suitable strategy for the crawling. First we discuss a variation of Focused Crawling in the first subsection and then we go to the problem of Indian language identification.

3.4.1 Using Focused Crawling Mechanism

Somboonviwat et al. [5] proposed a method based on Focused Crawler by Chakrabarti et al. [1] and they performed experiments on Japanese and Thai datasets. But the focused crawler is based on the fact that there is topical locality among the web pages which means the web pages belonging to a topic are located at a closer distance to each other. So in order to employ this method, the web should have language locality property. The authors mention that there is a substantial evidence for this property. Page relevance is given by the fact that whether the page belongs to the desired language. For detecting the language, the following methods can be used:

- **Character encoding:** One method is to use character encoding scheme present in the web page META tag.
- **Character distribution:** Analyzing the distribution of bytes of web page and guessing the character encoding schemes is another method.

Two strategies are proposed for assigning priority to a URL: Simple and Limited Distance.

3.4.1.1 Simple Strategy

Like the original focused crawler, two modes can be employed:

- **Hard Focus:** The relevance of a page is 1 if it is of the desired language and 0 if not. The extracted URLs are pushed to the Frontier only if the relevance score is 1 otherwise the crawl along that path is pruned.
- **Soft Focus:** The relevance can be any real value between 0 and 1. It is used to assign priority level to extracted URLs.

3.4.1.2 Limited Distance Strategy

The crawler proceeds along the path until N number of irrelevant pages are encountered consecutively. Again two variations exist:

- **Prioritized:** The URL is assigned priority which is inversely proportional to its distance from the last relevant page in that path.
- **Non-prioritized:** All URLs are assigned equal priority.

3.4.2 Detecting Indian Languages

Pingali et al [6] developed a search engine called WebKhoj for Indian language content which addresses the problem of multiple encodings of Indian languages of which most of them are proprietary and need some standardization. Indic scripts are phonetic in nature. When a vowel is added to a consonant, it becomes a syllable. Also there are compound syllables called ligatures. For example, consider 'tri' in triangle which contains three sounds. In most of the Indic scripts, it is a single compound character where as in English there are three characters. Very large number of glyphs are possible by combining multiple consonants and vowels. But the size of address space is only 256. The main idea is to transliterate the proprietary encodings into a standard encoding(UTF-8) and accept the user queries in the same encoding.

This is a semi automatic process. First the text from the web pages collected is divided into words using a word tokenizer. The algorithm lists all possible word beginning byte sequences in both source and target encodings. Each word is scanned from left to write until one word beginner appears. The word is tokenized at this point. The frequency of each byte sequence is calculated. The potential characters are sorted and and shown to the user. The user maps the equivalent matchings in source and target encodings. During crawling, if a page is found out to be of encoding other than UTF-8, the appropriate table is used for transcoding. The maximum byte sequence in the table that can be matched is used for transcoding.

3.5 Summary

In this chapter, we have discussed focused crawling in detail. We described two methods of Focused Crawling, the first one uses hierarchical structure of web and then we saw some shortcomings of this mechanism and discussed an alternative method which uses Context Graphs modeling the neighbors of relevant pages. Then we discussed two webpage classification techniques: the first one is a hierarchical classification technique using SVM and the second one is a positive example based learning technique which also uses SVM. Then we went on to discuss an application of focused crawling, language specific crawling. We describe a variant of Focused Crawler and then recognized the problem of detecting Indian languages due to proprietary encodings, and described a method to solve this problem.

Chapter 4

Probabilistic Ranking Methods

In the first section, we are going to discuss about Probabilistic Ranking Principle(PRP) and some models based upon it. In the next subsection we will see some shortcomings of PRP and discuss a variant of PRP for Interactive Information Retrieval proposed by Fuhr[10]. In the next section we will discuss another variant of PRP proposed by Chen et al.[7] which performs better when the purpose is to have atleast one relevant document.

4.1 Probabilistic Ranking Principle

Let R_{dq} be an indicator random variable which says whether a document d is relevant with respect to query q. Then it makes sense to order the documents according to decreasing values of $P(R_{dq} = 1|d, q)$. From this, PRP follows. "If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data."

There are two variants of PRP:

- **1/0 loss case:** This is the fundamental version of PRP where retrieval/selection costs are not taken into consideration. The documents are ranked by decreasing order of $P(R = 1|d, q)$ and the top k documents are returned. If only a set of relevant documents is required without any ranking, the documents with $P(R = 1|d, q) > P(R = 0|d, q)$ are returned.

- **Retrieval costs:** Let C_1 be the cost of retrieving relevant document and C_0 be that of non-relevant document. The documents are ranked by increasing order of $C_1P(R=1|d) + C_0P(R=0|d)$ and the top k documents are returned.

The following subsections give two ranking models which are based on PRP.

4.1.1 Binary Independence Model

The term binary corresponds to the fact that documents and queries are represented by binary term incidence vectors. Document d is represented by $\overline{d} = (x_1, x_2, .., x_n)$ where x_t=1 if term t is present in the document and 0 otherwise. The query is also represented in the same manner. The term independence corresponds to the fact that the term occurrences are assumed to be independent of one another. Also the relevance of one document is independent from the relevance of another document. This assumption is far from true especially in the case of Interactive Information Retrieval which we discuss in the next section. Using Bayes Rule, we have

$$P(R=1|\overline{x},\overline{q}) = \frac{P(\overline{x}|R=1,\overline{q})P(R=1|\overline{q})}{P(\overline{x}|\overline{q})} \tag{4.1}$$

After making Naive Bayes Conditional Independence assumption and estimating the probabilities by some measures which we give below and simplifying, we get the following formula which represents the relevance of a document d for a query $\overline{q}$. Logarithm is used because it simplifies the values and the ranking is not affected. The resulting quantity is called Retrieval Status Value(RSV).

$$RSV_d = \sum_{x_t=q_t=1} log \frac{s/(S-s)}{(df_t-s)/((N-df_t)-(S-s))} \tag{4.2}$$

Here N is the total number of documents, S is the number of relevant documents, s is the number of relevant documents in which the term t is present, df_t is the total number of documents in which the term t is present.To avoid the possibility of zero inside the logarithm, 1/2 is added to each term.

4.1.2 Non Binary Model : Okapi BM25

Binary Independence Model works well for short text files and catalogs but for large documents, unless we take into account the term frequency and document length, the results will not be satisfactory. Okapi BM25 is a probabilistic model which is built taking into account these terms. It considers the document as a bag of words but the

independence assumption is still present. We will approach the formula through a series of steps without dwelling too much into theory. The first and the simplest possibility is that the score for document d is sum of idf scores of each word in d.

$$RSV_d = \sum_{t \in q} log \frac{N}{df_t} \tag{4.3}$$

By taking into frequency of each term and document length into account, we can improve the above equation to:

$$RSV_d = \sum_{t \in q} [log \frac{N}{df_t}] \frac{(k_1 + 1) tf_{td}}{tf_{td} + k_1(1 - b) + b(L_d / L_{ave})} \tag{4.4}$$

In addition to the common variables with the Binary Independence Model from the previous section, tf_{td} is the frequency of term t in document d, L_d is the length of the document, L_{ave} is the average length over all the documents, k_1 is a parameter deciding the document term frequency scaling. b is a parameter that determines the document length scaling.

4.2 PRP for IIR

There are some assumptions in the basic PRP which do not hold in real life. For example, the assumption that the relevance of one document is independent of relevance of some other document is clearly not the case in Interactive Information Retrieval if one document's content is almost the same as another. If the first relevant document appears before the second almost duplicate document in the ranked list, then the second document loses its relevance. Also the information needs of the user are not static throughout a session of search as they change in reaction to the information a user has seen already. The goal is to fulfill the following requirements:

- All the interactions of the user with the IR system should be taken into consideration. Eg: Browsing through the document, coming back to the search results after reading a document etc.
- Different costs should be alloted to different types of effort put in by the user. For example, selecting a suggestion requires less effort of the user than making the user to expand the query.
- Changes in information needs of the user when a relevant document is found should be taken into account.

4.2.1 Cost Model

Fuhr[10] developed PRP for IIR by using a cost model. IIR is modeled as the user moving from one situation to another. In each situation, the user is presented a list of binary choices. $C_i = \{c_{i,1}, c_{i,2}, ..., c_{i,n_i}\}$. The first positive decision taken by the user changes the situation. The following table contains the notations used in forming PRP for IIR.

Notation	Definition
S	set of situations
s_i	situation
C_i	set of choices in situation s_i
n_i	$\|C_i\|$ (number of choices)
c_{ij}	single choice
p_{ij}	probability that choice c_{ij} is accepted
q_{ij}	probability that acceptance of c_{ij} is correct
e_{ij}	user effort for evaluating c_{ij}
b_{ij}	benefit from accepting c_{ij}
g_{ij}	correction effort if c_{ij} was chosen erroneously
a_{ij}	average benefit if c_{ij} is accepted

The expected benefit of choice c_{ij} :

$$E(c_{ij}) = e_{ij} + p_{ij}(q_{ij}b_{ij} + (1 - q_{ij})g_{ij}) \tag{4.5}$$

In order to maximise the expected benefit of the situation,we have to consider the whole set of choices. For brevity we denote average benefit of accepting c_{ij} which is $q_{ij}b_{ij} + (1 - q_{ij})g_{ij}$ by a_{ij}. Then $E(c_{ij}) = e_{ij} + p_{ij}a_{ij}$ By doing some calculations, we get the PRP for IIR, which says that we should rank choices c_{ij} by decreasing values of

$$a_{ij} + \frac{e_{ij}}{p_{ij}} \tag{4.6}$$

We can get the classical PRP from this by taking $e_{ij} = C < 0$ (the cost for reading a document) and $a_{ij} = \overline{C}$ (the benefit of a relevant document). Substituting these values we get $p_{ij} \geq p_{i,j+1}$. Thus we get classical PRP where documents are ranked by decreasing values of their probability of relevance.

4.3 Probabilistic Model for Fewer Relevant Documents

Chen et al. [7] argue that PRP is not optimal when the goal is to find at least one relevant document in the top n documents or when we want to retrieve documents relevant to different interpretations of the query in top n docs. %no metric is the fraction of test queries for which no relevant results are present in top n documents. This metric captures the first goal. Expected Metric Principle(EMP) is the foundation for the proposed algorithm. This principle states that, in a probabilistic context, one should directly optimize for the expected value of the metric of interest. This principle is applied to %no metric in the algorithm. Though this is a computationally intractable problem, a greedy algorithm is proposed which approximates the objectives.

The %no metric is generalized with a class of binary metrics, k-call at n. Given a ranked list, k-call at n is 1 if at least k of the top n documents returned by the retrieval system for the given query are deemed relevant. Our aim is to maximize the k-call at n metric by maximizing the probability that we find k relevant documents among the first n documents retrieved. If d_i is the i^{th} document in the ranked result set, r_i is a binary variable which is equal to 1 if d_i is relevant, then the objective function which should be maximized is: $P(\text{atleast k of } r_0, ..., r_{n-1} | d_0, ..., d_{n-1})$

4.3.1 Greedy Approach

We show the approach for k=1 and k=n.

- **k=1** The first document d_0 is that which maximizes $P(r_0|d_0)$. The second document d_1 is selected so as to maximize $P(r_0 \cup r_1 | d_0, d_1)$. When we simplify we get only one term which depends on d_1 which is $P(r_1 | \neg r_0, d_0, d_1)$ and other terms are independent of d_1. Similarly, d_i is selected by choosing the document which maximises $P(r_i | \neg r_0, ..., \neg r_{i-1}, d_0, ..., d_i)$
- **k=n** For this case, we find that d_i is selected by choosing the document which maximises $P(r_i | r_0, ..., r_{i-1}, d_0, ..., d_i)$.

In some cases of long queries, it has been shown that this beats PRP in retrieving relevant documents. This is also successful in retrieving documents relevant to distinct interpretations of a query.

4.4 Summary

In this chapter, we have discussed in detail the Probabilistic Ranking Principle(PRP) and some models based upon it. Then we saw some shortcomings and invalid assumptions of PRP in certain conditions and discussed some variants of PRP which can be applied in those conditions. In particular, we described a variant of PRP which can be applied during Interactive Information Retrieval and another variant of PRP which can be applied when the purpose is to retrieve at least one relevant document.

Chapter 5

Personalised Information Retrieval

5.1 Introduction

Personalised systems are user-centric, not in the sense of generic user but unique user. We can also say that these systems adapt to the user, and give information views personal to that user. Web based personalised IR systems learn about the user implicitly from their previous actions or interactions and also explicitly from the information given by the user. Personalisation can be divided into three categories:

- **Profile based:** First the system asks the user to fill a profile in which information of user interests and activities is taken which is used in the personalisation.
- **Collaboration based:** Based on a user's actions with the system, the system detects users which are performing similar actions and present the results based on the other actions of those similar users. Collaboration based approaches are mostly used in recommender systems. For example, we can see some suggestions on Amazon.com which show the things bought by the customers who bought the same things as us.
- **Learning based:** The system uses user's interactions with the system like previous searches and learns some models which are unique to a user. In profile based systems, the information regarding the user is taken explicitly, whereas here it is taken implicitly and the model is continuously evolving.

In the next section, we see a method which uses data present in client system for performing re-ranking of results locally, proposed by Teevan et al.[11]. In the second section, we

see how can personalisation be performed at retrieval stage itself using ontology-based framework in which we do not worry about how exactly we gather information about the user. This is proposed by Castells et al.[12].

5.2 Automated Analysis of Interests and Activities

The information used for modeling the user's interests and activities is taken from previously issued queries and previously visited web pages and the documents and emails present in the user's desktop. This information is used for re-ranking the search results locally. Local re-ranking has the following advantages: Can use computationally intensive procedures for re-ranking as the results to be re-ranked are very few, need not send personal information to a search engine with which some users might be uncomfortable. Modified BM25 scheme is used and also relevance feedback is incorporated based on the documents marked as relevant by the user.

The given query is expanded using the relevant documents and the modified BM25 is applied on the web pages corresponding to the results obtained. The BM25 in its simplest form is nothing but the weighted sum of the term frequencies. The term weight when there is no relevance information is: $w_i = log\frac{N}{n_i}$. Let N be the number of documents in

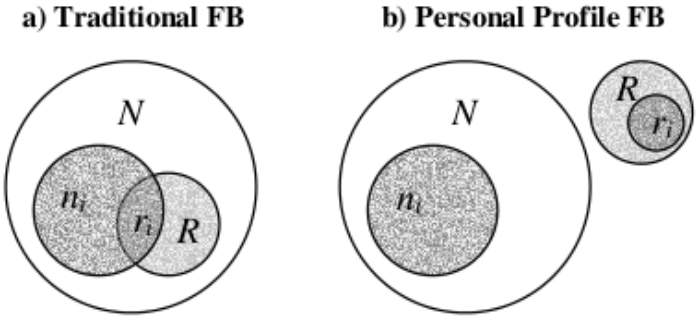

FIGURE 5.1: Set representation of feedback and the corpus

the corpus, and n_i be the number of documents in the corpus that contain the term i. Let us assume we have some relevance feedback information. Let R be the number of relevant documents and let r_i be the number of documents containing term t_i. In this case, the set representation is given in 5.1(b). Then the weight can be modified to the following formula after accounting for the cases of zero by adding 0.5. to each term.

$$w_i = log\frac{(r_i + 0.5)(N - n_i + 0.5)}{(n_i + 0.5)(R - r_i + 0.5)} \tag{5.1}$$

- **User representation:** User is represented by an index of personal content taken from the web pages viewed, emails viewed or sent, documents stored on the client

machine. The most straightforward way of using this is to consider every document in this index as relevant to the user. Let R be the number of documents in the index and r_i be the number of documents in R containing the term i. We can experiment with different subsets of index like including only the pages viewed recently by the user or including only web pages.

- **Corpus representation:** N is the number of documents in the web and n_i is the number of documents containing the term i. Another alternative is to set N as the number of documents containing the original query terms and n_i is the number of documents containing the query terms and the term i. We obtain estimates of these parameters by issuing one word queries to the search engine and using the number of results reported by it.

- **Document and Query representation:** Since accessing the full text of a document corresponding to a web search result takes considerable amount of time, the snippets returned with the results can be used for representing the whole document. The query is expanded by including the terms present nearer to the query in the relevant documents.

5.3 Ontology Based Framework

Ontologies are effective in reducing ambiguity and help to express user interests. Castells et al. [12] developed a framework which exploits the advantages of ontologies for personalised information retrieval. If personalisation goes wrong, the results may become completely irrelevant (far worse than the original results) making the user opt out of personalisation. So it is essential to identify in what situations it is or it is not appropriate to use personalisation methods.

First we describe how IR can be done in ontology based framework without personalisation. Let D be the corpus of documents annotated by concepts from an ontology O. Document is represented by vector d of concept weights. For each concept $x \in O$ annotating d, d_x is the importance of x in document d. It can be calculated by using TF-IDF algorithm:

$$d_x = \frac{freq_{x,d}}{max_y freq_{y,d}} log \frac{|D|}{n_x} \tag{5.2}$$

where $freq_{x,d}$ is the number of occurrences of x in d, $max_y freq_{y,d}$ is the frequency of the most repeated instance in d, and n_x is the number of documents annotated by x. For this a mapping from concepts to the keywords representing the mapping must be available. Similar is the representation of the query. Then cosine similarity between the query and the document is used as the relevance score for ranking the documents.

Now we come to personalisation aspect. The preferences of user are denoted by a vector $u \in [0,1]^{|O|}$ of concept weights. First the documents which are marked as relevant during the feedback sessions are taken and the concepts annotating the documents are listed out. All these concepts may not be relevant to the user. In order to find out the weightage given to each concepts, fuzzy clustering is performed on the concepts. The concepts which are matching with few other concepts form clusters of low cardinality. The cardinality of clusters is used in calculating the weightage of each concept as follows:

$$u_x = \sum_{t \in T} \mu(x,t) L(t) K(t) \tag{5.3}$$

for each $x \in O$ where $t \in T$, $\mu(x,t)$ denotes the degree of membership of the concept x to the cluster t, and $K(t) = \bigcap_{d \in T} R(d,t)$.

The score of a document d with respect to a query q for a user u is given by

$$score(d,q,u) = \lambda.prm(d,u) + (1-\lambda).sim(d,q) \tag{5.4}$$

where prm(d,u) is the cosine similarity between the vector representations of document and the user and sim(d,q) is the cosine similarity document and query. $\lambda \in [0,1]$ indicates the degree of personalisation.

The degree of personalisation should be low if there is a high certainty that the results obtained without personalisation are relevant, otherwise the degree should be high. The vagueness in the user requests and system responses are used to approximate this uncertainty. This vagueness is defined as a function of three measures: specificity of formal query, query result set and final result set. This function should be monotonically increasing with respect to any of the three measures. A query with many conditions and few variables is considered to be more specific. A query that is satisfied by less ontological instances is taken to be more specific. If fewer results are returned in a search without personalisation, it is considered to be more specific.

5.4 Summary

In this chapter, we discussed the types of Personalised IR and described two methods which are Learning Based. The first method uses data present in client system for performing re-ranking of results locally and the second method uses ontology-based framework in which personalisation is performed at retrieval stage itself.

Chapter 6

Evaluation Methods

In this chapter, we will discuss different evaluation methods which are used as a standard in various Information Retrieval systems. The effectiveness of an IR system are test data dependent. If one uses some custom test data and another uses some other test data for testing the effectiveness of their respective IR systems, we cannot be sure which one performs better in general. For this, there is a need for some standard test data collections which can be used by everyone. A test collection must consist of the following: a set of documents, a set of queries representing the information needs, information about the relevance of each document with respect to each and every query.

We will see some standard test collections after which we will discuss some standard measures for unranked and ranked retrieval sets.

6.1 Standard Test Collections

- **Cranfield:** This is an old collection of documents but it is very small and can be used for only preliminary experiments. This consists of of 1398 abstracts of aerodynamic articles, 225 queries, and relevance information for each query,document pair.
- **Text REtrieval Conference (TREC):** It is organised as a series of workshops and its purpose is to support IR research by providing necessary infrastructure for evaluation. This infrastructure includes test collections which are used as a standard in IR community. A separate track exists for each research area and new tracks are added as new research areas get identified. For example, these are some of the tracks in TREC 2013:

  - **Contextual Suggestion Track:** Investigates search techniques focusing on context and user interests.
  - **Knowledge Base Acceleration Track:** Develops techniques to improve the efficiency of human knowledge base curators by suggesting extensions to the existing KB.
  - **Microblog track:** Investigates information seeking behaviors in microblogging environments.

- **NII Testbeds and Community for Information access Research (NTCIR):** These collections are similar in size to the collections provided by TREC but for East Asian languages and they are focused on cross language information retrieval.

- **Conference and Labs of the Evaluation Forum(CLEF):** This forum provided test collections for evaluation of IR systems for European languages and cross language information retrieval among these languages.

- **Forum for Information Retrieval Evaluation(FIRE):** Its main aim is to encourage research in South Asian language Information Access technologies by providing reusable large-scale test collections for cross lingual IR experiments.

6.2 Unranked Retrieval Sets

We will first see how the effectiveness of an IR system which returns a set of documents for a query without ranking is measured. Precision and Recall are the most important measures in this case. Precision is defined as the ratio of the number of relevant documents retrieved to the total number of documents retrieved. Recall is defined as the ratio of the number of relevant documents retrieved to the total number of relevant documents in the corpus. In general, the number of relevant documents are a very tiny fraction of the whole corpus. So, in good IR systems, precision and recall trade-off against each other. This is because as we retrieve more and more documents recall may increase but not decrease. But precision will be high initially in a good IR system and as more and more documents are retrieved, almost all of them will be irrelevant resulting in a decrease in precision.

F measure takes into account both precision and recall and it is the weighted harmonic mean of precision and recall.

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} where \beta^2 = \frac{1-\alpha}{\alpha} \tag{6.1}$$

For equal weightage set $\beta = 1$. Harmonic mean is used instead of geometric or arithmetic mean because if we use AM, and if an IR system returns all the documents, then recall is 1, so $F_{\beta=1}$ is greater than or equal to $1/2$ always which we can see that it is not desirable. Harmonic mean is closer to the minimum of the two when the difference between the two measures is large.

6.3 Ranked Retrieval Results

The measures given in the previous section do not give any idea about how many relevant documents are at the top compared to irrelevant documents. For example consider precision for a set of k retrieved documents. The precision when all the relevant documents occur after the irrelevant documents in the list, is the same as the precision when relevant documents occur before the irrelevant documents.

- **MAP:** One of the most important measures which captures this is Mean Average Precision(MAP). In verbal terms, this is average precision over all the test queries. Average precision for a single query is defined as average of the precision of the set of retrieved documents after each relevant document is retrieved. MAP can be defined as:

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}) \tag{6.2}$$

  Here Q is the set of queries. Let $d_1, d_2, ..., d_{m_j}$ be the relevant documents with respect to query q_j. After a document d_k is retrieved, let the set of retrieved documents be R_{jk}.

- **Precision at k:** MAP averages out the precision at all recall levels. But in web search, we can observe that we do not go beyond first 2 or 3 pages of search results (i.e. 20 to 30 results). So it makes sense to calculate precision for top k documents for multiple values of k below 30.

- **R-precision:** Let there be $|R|$ relevant documents for a query. Consider the top $|R|$ documents retrieved for that query which contains $|r|$ relevant documents. At this point, the precision is essentially equal to recall.

- **NDCG:** Normalised Discounted Cumulative Gain (NDCG) is another one which is mainly used in the cases where relevance of a document with respect to a query can be any real value between 0 and 1. It is evaluated for top k search results. If R(j,d) is the relevance score of document d w.r.t. query j, then NDCG can be

defined as:

$$NDCG(Q,k) = \frac{1}{|Q|}\sum_{j=1}^{|Q|} Z_k \sum_{m=1}^{k} \frac{2^{R(j,m)}-1}{log(m+1)} \tag{6.3}$$

Here Z_k is the normalisation factor calculated so that when the ranking is perfect, the NDCG value at k is 1.

6.4 Summary

In this chapter, we have discussed different test data collections which are used as a standard in IR community. Then we went on to describe some evaluation measures which are used frequently to measure the effectiveness of unranked retrieval sets like precision, recall, F-measure. We also did the same for ranked retrieval results, defining some measures like Mean Average Precision, Precision at k, R-precision and NDCG.

Chapter 7

Experiments

Apache Nutch is an open source web crawler which is extensible and scalable. It can run on a single machine or can run in a distributed computing environment, for example in a Hadoop cluster. In small scale search engines and academia, Nutch is highly used along with Apache Lucene for indexing and retrieving ranked results. Since the future work will be done on Nutch, I have worked on it to familiarise with the code flow. In the first section, we will briefly describe the functioning of Nutch. Then we will be discussing some experiments in which we perform Telugu language specific crawling on Nutch.

7.1 Structure of Nutch

Nutch is written in Java. Since we have to work on the source code of Nutch, it would be very convenient to work on a Java IDE. We chose Eclipse because it contains better plugins than Netbeans. The first and foremost thing to do is to download the source code of Nutch and run it as an Eclipse Project. The whole installation procedure of Nutch in Eclipse has been clearly written step-by-step in this site: Link. If we ant build the project, we can find a Nutch installation in runtime/local directory in the root of the source tree. You can use the official tutorial to know how to perform basic crawl. The following data are stored when we perform a Nutch crawl:

- **Crawl database (crawldb):** Contains information about evry URL known to Nutch. For example, the information about whether a known URL is fetched or not and if it is, the time it was fetched etc.
- **Link database (linkdb):** For each URL, linkdb maintains a list of known URLs pointing to that URL and also the anchor text of the link.

- **Segments:** Multiple segments will form in the segments directory as the crawl depth increases. Each one corresponding to a single iteration of the crawl. The following are the contents of a segment:
  - crawl_generate: set of URLs to be fetched.
  - crawl_fetch: fetch status of each URL.
  - parse_text: parsed text of fetched URLs.
  - parse_data: outlinks and metadata of fetched URLs.
  - crawl_parse: outlink URLs used to update crawldb.

Crawl.java is the starting point which contains the main function which calls all the components of the crawler in the order of process of crawling: Injector, Generator, Fetcher, Parser, CrawlDbUpdater are the most important ones. The following are the outlines of the functions performed by each component.

- **Injector:** Present in Injector.java. Takes the input as crawldb directory and directory containing list of seed URLs and performs applies normalisers and filters on the URLs and initialises crawldb.
- **Generator:** Present in Generator.java. Analyses the crawldb and generates a new segment with a sub directory crawl_generate containing the info about the URLs to be fetched.
- **Fetcher:** Present in Fetcher.java. Takes the segment directory as input and tries to fetch URLs present in crawl_fetch.
- **Parser:** Present in ParseSegment.java. Parses the obtained content with all the suitable parsers and generates content in parse_data, parse_text.
- **CrawlDb Updater:** Present CrawlDb.java. Using the information about the URLs fetched and which or not in the last segment, it updates the crawldb.

7.2 Language Specific Crawling

In order to crawl the pages of only a specific language, we have to add a parse filter plugin which is evoked while parsing the fetched page. This plugin should detect the language of the page and decide whether to keep this page or discard it. First of all, a Nutch plugin is nothing but an extension of one or more extension points. There are certain extension points in Nutch, each of which is an interface that must be implemented by an extension. In our case, we have to write a plugin which provides an extension for the

extension point HtmlParseFilter. There is already a LanguageIdentifier plugin which detects the language and adds it to the meta data of the URL. We modified this plugin to return null if the detected language is not Telugu. This results in pruning the crawl at that stage and the out links from that web page are not added to the to-be-fetched list for next iteration.

7.3 Results and Observations

The seed URL was given as http://eenadu.net/. This is an online telugu newspaper. This resulted in ending the crawl immediately due to the language being detected as some other language. This lead to no outlinks being added to the next iteration. The same thing happened with all the Telugu URLs we tried. We came to know that the Telugu language is being detected as Lithuanian language most of the times. The reasons are still to be investigated. So we tried to use an external library Indian Language Identifier for this purpose but due to some limitations with Apache Maven we were not able to add an external library to the Eclipse Project. There are some complicated turnarounds for this which did not work. We are looking for alternatives for this. We tested Language Specific Crawling for French websites with seed URL as http://www.inria.fr/. This worked as expected as the language was detected as French every time the page was in French and the crawl was pruned whenever an English page was encountered.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this report, we have discussed the architecture of crawling ,focused crawling, probabilistic ranking models, personalised IR and evaluation methods for IR. In Crawling, we have discussed the architecture of a crawler and saw how it can be extended to distributed crawler. In Focused crawling, we have seen two types of focused crawler one using hierarchical structure of the web and the other modeling the context around relevant pages. Also we discussed two web page classification techniques one based on hierarchical structure and the other using only positive examples to train the classifier. Both of the methods used SVM. Then we went on to discuss Language Specific Crawling and saw how a slight variation of Focused Crawler can be used to implement it. In Probabilistic Ranking Models, we have discussed the Probabilistic Ranking Principle and some models based on it. Also we have seen some shortcomings of PRP in certain situations and discussed some variations of it. In Personalised IR, we discussed two methods: One using a variation of BM25 to rank the results locally and the other using ontology based framework to rank the results at the retrieval stage itself. In Evaluation methods, we discussed various test collections which are used as a standard in IR community. Then we went on to discuss various measures which show the effectiveness of a retrieval system in both the cases of ranked and unranked retrieval sets. Then we discussed some experiments performed with Apache Nutch to do language specific crawling.

8.2 Future Work

- Short term:
  - Perform language specific crawling for Telugu language. Also mix topic focused and language specific crawling for example tourism related pages in Hindi language.
- Mid term:
  - Optimising crawl resources and configurations given in-links, out-links, fetch times, parse times of a carefully chosen sample of web pages can be a very useful contribution.
- Long term:
  - Focused Crawling is not used widely in practice, inspite of its advantages. This may be because modern web pages do not belong to a single topic. Improvements in Focused Crawling is one possible area of future work.
  - Personalised IR in small scale search engines is also difficult due to lack of user information and increase in processing time. There is a need for efficient and new methods in this area.

Bibliography

[1] Soumen Chakrabarti, Martin Van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11): 1623–1640, 1999.

[2] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C Lee Giles, Marco Gori, et al. Focused crawling using context graphs. In *VLDB*, pages 527–534, 2000.

[3] Susan Dumais and Hao Chen. Hierarchical classification of web content. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 256–263. ACM, 2000.

[4] Hwanjo Yu, Jiawei Han, and Kevin Chen-Chuan Chang. Pebl: positive example based learning for web page classification using svm. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 239–248. ACM, 2002.

[5] Kulwadee Somboonviwat, Masaru Kitsuregawa, and Takayuki Tamura. Simulation study of language specific web crawling. In *Data Engineering Workshops, 2005. 21st International Conference on*, pages 1254–1254. IEEE, 2005.

[6] Prasad Pingali, Jagadeesh Jagarlamudi, and Vasudeva Varma. Webkhoj: Indian language ir from multiple character encodings. In *Proceedings of the 15th international conference on World Wide Web*, pages 801–809. ACM, 2006.

[7] Harr Chen and David R Karger. Less is more: probabilistic models for retrieving fewer relevant documents. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 429–436. ACM, 2006.

[8] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

[9] Christopher Olston and Marc Najork. Web crawling. *Foundations and Trends in Information Retrieval*, 4(3):175–246, 2010.

[10] Norbert Fuhr. A probability ranking principle for interactive information retrieval. *Information Retrieval*, 11(3):251–265, 2008.

[11] Jaime Teevan, Susan T Dumais, and Eric Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 449–456. ACM, 2005.

[12] Pablo Castells, Miriam Fernández, David Vallet, Phivos Mylonas, and Yannis Avrithis. Self-tuning personalized information retrieval in an ontology-based framework. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, pages 977–986. Springer, 2005.