Project Report

Faiyaz Sundrani and Mohammed Ilyas Habeeb

Web Search Engines,
New York University, USA
fs1459@nyu.edu and mih278@nyu.edu

Abstract:

In this paper, we present focused crawler using Machine Learning, a prototype of focused crawler which makes use of Machine Learning to predict the next best page to be crawled. Our crawler is designed to crawl the web efficiently and produce much more satisfying search results than existing crawlers. To engineer a crawler is a difficult task. Crawlers can crawl millions of web pages, producing results for millions of queries every day. This paper provides an in-depth description of our focused crawler using Machine Learning and we compare the results of each of the Machine Learning model we used. Apart from crawling, we will be ignoring pages with are not text page. Also, we look at the how to update the promise of each page by applying ML after each page crawled.

Keywords:

Focused Crawler, crawler, Machine Learning

1.Introduction

As the web has been immersed in our day to lives, the search engine on webs has become a necessity for humans, getting result related to the query has become more and more challenging as the number of pages on the web has increased. Google has become the main search engine for users and for any search engine it has become a task to crawl the pages regularly so that the users get up to date results for the queries. A daunting task is to crawl all the web pages regularly. A common practice is to crawl pages which mostly search regularly. The other pages which are hardly seen by the user can be crawled time to time basis to keep it updated. One of the big tasks is to crawl relevant pages which are likely to be seen by the user or likely to be the page the user will be looking for. We have used a Machine Learning algorithm which focuses on finding the best page that might be related to the user query without crawling the web page first. Our aim is to crawl nearly 5000 pages which will be relevant to a user query.

As we crawl the pages, the promise will be updated using the ML model which will predict the promise of a URL without opening the page. A crawler can also validate hyperlinks and HTML code.

1.1 Crawlers

A crawler starts to crawl with a list of URLs given. Here we will be using Google's top results for the start of our URL list which is known as seeds. All the initial links will be given a score and accordingly as the crawler starts to visit these URLs, it is going to identify all the hyperlinks all will add it to a list to be visited later. This list is visited regularly according to out crawling policy. The crawler is going to archive the URL links along with other factors like score, etc. As there are billions of pages on the web crawler can only crawl limited web pages in particular time, so we will optimize our crawler accordingly to crawl only the most relevant links. One of the major tasks is to avoid retrieving duplicate contents by the crawler.

1.2 Focused Crawlers

Focused Crawler crawls web pages that follow a specific policy by carefully prioritizing the URLs which are needed to be visited next. Some focused crawlers are simple or are based to satisfy certain properties, like a crawler will only crawl .xyz domains or a crawler might only crawl health-related websites, these specific conditions reduce the crawlers resources while fetching the web pages, as it does not need to crawl all pages continuously but only to a specific domain. The main task of a focused crawler is to predict that an unvisited page will be relevant before downloading the page. There are many ways to predict that a link is relevant or not. In the past years, the prediction was done on a simple basis like if the query term existed in the URL or not. But as web grew, Machine Learning techniques have been applied to the crawler where the relevance of a page is determined by different machine learning models. Simple crawling techniques have been effective for short crawls but more sophisticated techniques such as reinforcement learning are proved to be giving best performance over longer crawls. Reinforcement learning has been one of the latest and most effective strategies to be implemented on focused crawling where online based classification algorithm is used in combination with a bandit based selection strategy to efficiently crawl pages.

Focused crawlers performance mainly depends on the richness of links in the specific topic that is being searched and focused crawling usually relies on a general web search engine for providing starting points that is seeds. Seed selection is important for focused crawlers and significantly will influence the crawling efficiency. One of the

popular strategies in whitelist strategy where the crawler starts to crawl from the list of high-quality seed and limits the crawling scope to domains of URLs. The seeds are selected based on a list of URL candidates which are accumulated over a sufficiently long period of general web crawling.

1.3 Design Goals

Experiment Methodology:

We aim to conduct this experiment on both large topics and rare topics. For each topic, we crawl 5000 documents. We will crawl the pages according to two strategies:

1. Without Machine Learning
2. With Machine Learning

1.3.1 Without Machine Learning Methodology:

In this methodology, we crawl the documents without applying any Machine Learning. Specifically, we estimate the promise of each link by calculating:

1. The ratio of the number of occurrences of the query word in the hyperlink and the total number of words in the hyperlink
2. The ratio of the number of occurrences of the query word in the hyperlink text and the total number of words in the hyperlink text
3. The ratio of the number of occurrences of the synonyms of the query word in the hyperlink and the total number of words in the hyperlink
4. The ratio of the number of occurrences of the synonyms of the query word in the hyperlink text and the total number of words in the hyperlink.
5. Average cosine measure of all the parents of the URL

For 3rd and 4th, each of the ratios of the synonym of the query word is added and multiplied by 0.5. This is done to signify that the synonyms are only half as important as the query word itself. After that, we add the sum of 1, 2, 3, 4 to 5th to compute the promise. The crawler then crawls the page with the highest promise.

Furthermore, while evaluating all the hyperlinks (children links) of a parent link, we estimate the promise of each hyperlink and filter out the top 10 children links and discard the rest of the hyperlinks.

1.3.2 With Machine Learning Methodology:

In this methodology, we consider the following features:

1. Average Cosine Measure of all the parents of the URL,
2. The frequency of the query word in the hyperlink,
3. The frequency of the query word in the hyperlink text,
4. Sum of the frequency of all the synonyms of the query word in the hyperlink,
5. Sum of the frequency of all the synonyms of the query word in the hyperlink text,

Based on the above features, we aim to estimate the cosine measure of the URL.

X = [x_1, x_2, x_3, x_4, x_5] where:

x_1 = Average Cosine Measure of all the parents of the URL,
x_2 = Frequency of the query word in the hyperlink,
x_3 = Frequency of the query word in the hyperlink text,
x_4 = Sum of the frequency of all the synonyms of the query word in the hyperlink, and
x_5 = Sum of the frequency of all the synonyms of the query word in the hyperlink text

and y = Actual Cosine Measure of the URL

We aim to estimate y based on the features defined in X. Since y is a continuous value, we model this as a supervised Machine Learning regression problem.

1.3.3 Machine Learning Techniques

1.3.3.1 Linear Regression:

Linear regression is helpful in finding a relationship between two continuous variables.One variable is known as an independent variable and other is known as a dependent variable. Linear regression looks for a statistical relationship not deterministic. The relationship between the two variables is said to be deterministic if one variable can be accurately expressed by the other. For example, using temperature in degree Celsius it is possible to accurately predict Fahrenheit. Statistical relationship is not accurate in determining a relationship between two variables. For example, the relationship between height and weight. The main aim of using Linear Regression is to obtain a line that best fits the data. The best fit line is the one for which total prediction

error (all data points) are as small as possible. An error is a distance between the point to the regression line.
Example: Assume we have a dataset which consists of the relationship between 'number of hours studied' and 'marks obtained'. We design a model to predict marks given the number of hours studied. The regression line obtained is the one that will give a minimum error.

$$Y(pred) = b_0 + b_1 * x$$

Here the values of b0 and b1 must be selected in such a way that they reduce the error
If b1 is greater than 0 then predictor(x) and target(y) have a positive relationship. If x increases then y also increases.
If b1 is less than 0 then predictor(x) and target(y) have a negative relationship. If x increases then y decreases.
The value of b0 guarantee that residual have to mean zero. If there is no 'b0' term, then regression will be forced to pass over the origin. Both the regression coefficient and prediction will be biased.

1.3.3.2 SVR with Gaussian Kernel

Support Vector Regression:
It uses the same principles as the Support Vector Machine for classification. We try to fit the error within a certain threshold. They characterized by the usage of kernels, the absence of local minima, the sparseness of the solution and capacity control obtained by acting on the margin, or on a number of support vectors. Support Vector Machine can be applied not only to classification problems but also to the case of regression. It still contains all the features that are important that will characterize the maximum margin algorithm (it is a nonlinear function which is learned by linear learning machine mapping into a high dimensional kernel-induced feature space. The parameters that do not depend on the dimensionality of feature space control the capacity of the system.
In the same way as with classification approach, there is motivation to seek and optimize the generalization bounds given for regression. They relied on defining the loss function that ignores errors, which are situated within a certain distance of the true value. This type of function is often called epsilon intensive loss function.

Gaussian Kernel:
Below is the equation for a Gaussian with a one-dimensional input.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

x = input ,
μ = mean,
σ = standard deviation
The Gaussian function is based, first of all, on the Euclidean distance between the input vector and the prototype. The first thing you'll notice about the Euclidean distance is that it produces the inverse of the response we want–we want the neuron to produce it's the largest response when the input is equal to the prototype. We'll deal with that in the next section. The Gaussian function is based on the squared Euclidean distance. Note that squaring the Euclidean distance is the same as just removing the square root term. This leads to the $(x - \mu)^2$ term in the equation for the one dimensional Gaussian. For a one-dimensional input, the squared Euclidean distance is just the parabola $y = x^2$.

1.3.3.3 Regression Trees
Regression trees are used for variables that are dependent and that take continuous values, where prediction error is measured by the squared difference between observed and predicted values. Regression trees may be considered a variant of decision trees, designed to approximate real-valued functions, instead of being used for classification methods.

Binary recursive partition is the process known to build a regression tree. It is an iterative process that splits the data into partitions or branches and then continues splitting each partition into smaller groups as method moves up each branch.
Initially, the records present in training set (records that are pre-classified to determine the structure of tree) are grouped in the same partition. The algorithm then tries to allocate the data into the first two branches using every possible binary split on every level. Later, the algorithm selects the split that minimizes the sum of the squared deviations from the mean in the two separate partitions. This splitting rule is then applied to each of the new branches. This process continues until each node reaches a user-specified minimum node size and becomes a terminal node. (If the sum of squared deviations from the mean in a node is zero, then that node is considered a terminal node even if it has not reached the minimum size.)

Pruning the Tree
Since the tree is grown from the Training Set, a fully developed tree typically suffers from over-fitting (i.e., it is explaining random elements of the Training Set that are not likely to be features of the larger population). This over-fitting results in poor performance on real-life data. Therefore, the tree must be pruned using the Validation Set. XLMiner calculates the cost complexity factor at each step during the growth of the tree and decides the number of decision nodes in the pruned tree. The cost complexity

factor is the multiplicative factor that is applied to the size of the tree (measured by the number of terminal nodes).
The tree is pruned to minimize the sum of
1) the output variable variance in the validation data, taken one terminal node at a time; and
2) the product of the cost complexity factor and the number of terminal nodes. If the cost complexity factor is specified as zero, then pruning is simply finding the tree that performs best on validation data in terms of total terminal node variance. Larger values of the cost complexity factor result in smaller trees. Pruning is performed on a last-in-first-out basis, meaning the last grown node is the first to be subject to elimination.

To determine the best Machine Learning technique for our crawler, we conduct the experiment as follows:

1. Divide the dataset randomly into a training and a validation set.
2. Based on the training dataset, run 10-fold cross-validation to determine the optimal hyper-parameters for each of the three M.L techniques. This is done using a grid search on the logarithmically-spaced hyper-parameters.
3. Evaluate each of the three M.L technique with optimal hyper-parameters against the validation set.
4. The M.L technique that gives the highest R-2 Score is selected. R-2 is a statistical measure that indicates how much variance is explained by our model.

Machine Learning Technique	**R-2 Score**
Linear Regression	0.541
S.V.R with RBF Kernel	0.663
Regression Tree	0.582

Table 1: M.L Techniques with their Performance Metric R-2

Based on the above experiment, we determine **Support Vector Regressor with the rbf kernel** with C = 1, gamma = 1 as the best M.L technique for our crawler.

2. System Features

Our web focused crawler has multiple important features that will help us to produce high precision results. We will be using multiple checks so as to ignore certain types of links and make our result more defined.

2.1 Selection Policy

Currently, the web has billions of web pages and our strategy is to crawl only the relevant pages which will improve our crawler efficiency. We are using web pages based on Machine Learning Strategy as described above.

2.2 Restriction

One of the main features of our crawler is to avoid all MIME types. In order to request only HTML resources, we will be using mimetypes library of python. guess_type() function will help us to guess the type of page based on URL. Mostly the media flies like .mp4, .mp3, .jpg, .png and pdf files are our main to focus to ignore. We have included other types which are going to be ignored as we crawl. If the mimetype library fails to identify the type of page, we are manually finding out the page type using urllib library of python where we open the link and find the Content-Type: of the url which will give us the type of page. This strategy will help us to skip a lot of URLs. We will not be ignoring links with “?” in them as currently there are very few spider traps that may cause crawler to a downloaded infinite number of URIs from the web. Currently, many applications use URL rewriting to simplify the URL so that strategy would be unreliable.

2.3 Normalization

The main purpose of URL normalization is to avoid crawling same resources more than once. This process is used for modifying and standardizing URL in a consistent manner. Mainly a website might be written in uppercase/lowercase which will make the url crawl the page more than once, here URL normalization helps us to avoid crawling the same page again. There are many forms in which a URL can be written. A link has been attached below in references to help identify what all ways a URL can be duplicated which we will be preventing.We will be using the url_normalize library of python which always returns the URL in lowercases, eliminates ports, performs percent encoding, etc. This system feature will help us to skip the visiting of URLs multiple times and saving time.

2.4 Identification

As we crawl web pages, our main aim is to identify our crawler to the web page in order to avoid being seen as spam. As security is a main criteria for web pages, crawlers are a big threat to web pages unless they are from identified sources. It is important for the crawler to identify itself so that the Website administrator can contact the owner if needed. As crawler crawls a domain multiple times, making multiple requests to the domain makes the crawler overload the website with many requests. A web crawler identifies itself as User-agent field and in our case, its mostly * represented as to it. We check the robots.txt file of the website to visit and see the User-agent field to find out if we have the permission to crawl the web page or not. Websites contain robots.txt file to let the crawler know if it can crawl the web page or not. Major search engines use crawl delay functionality to let the website know the time of delay between the crawls to avoid overloading of the website. For those using Web crawlers for research purposes, a more detailed cost-benefit analysis is needed and ethical considerations should be taken into account when deciding where to crawl and how fast to crawl. We use urlparse library in Python to get the robots.txt file and find whether we have permission to crawl the web page or not. A crawler which does not check the permission before crawling may be seen as spam if it overloads the server and the IP address can be banned by the website administrator. Crawlers must check before crawling any website to avoid any spamming issues.

2.5 Canonicalization

One more major part of the crawler was to avoid duplicate content from website e.x abc.com and abc.com/index.html are the same page but have different URL. This canonical problem can be solved by the website itself by using the canonical tag in the link to let the crawler know of the URL. The canonical link element helps webmasters make clear to the search engines which page should be credited as the original. As for the crawler if the if multiple URLs contain the same result in the result set, the canonical link URL definitions will likely be incorporated to determine the original source of the content.

Example - <link rel="canonical" href="http://example.com/">

2.5 Cosine score

$$
\begin{aligned}
F(d, t_0, \ldots, t_{m-1}) &= \sum_{i=0}^{m-1} \frac{w(q, t_i) \cdot w(d, t_i)}{\sqrt{|d|}}, \\
w(q, t) &= \ln(1 + N/f_t), \text{and} \\
w(d, t) &= 1 + \ln f_{d,t},
\end{aligned}
$$

• N: the total number of documents in the collection
• ft : number of documents that contain term t
• fd,t: frequency of term t in document d
• |d|: length of document d

Relevance denotes how well a retrieved document or set of documents meets the information need of the user. Relevance is an important factor to judge the document based on how relevant it is to the query. Relevance score helps search engines to match the documents with use query. Calculating a relevance score is a challenging task as we have to take multiple considerations into point because document changes accordingly and sometimes the documents are small or contain the query term less frequently. To tackle such ups and downs we are implementing cosine score to calculate the relevance of the document. Cosine score is one of the good scoring methods to rank the documents on how relevant they are to the query. Cosine score provides
• a higher score for more occurrences of a word
• a higher score for rare words
• lower score for long documents

It takes into consideration multiple factors for making sure the document ranking is unbiased and to give the most related output for the given query. It considers multiple factors so that the words which occur rarely are given equal significance and other words are not ignored. It helps us rank relevant web pages and gives them score, it is unbiased for small documents too. Cosine factor is a vector ranking model, the

significance of using this model is to give the best score to the document without ignoring small documents or documents where the query words appear less frequently. The cosine function is thus used to rank the relevance of the documents for us to compare with promise.

2.7 Data Structure and Library

Our focused crawlers are optimized so that a large document collection can be crawled, searched with little cost. One of the major factors affecting the crawler is to choose the right data structure which updates the links and promise and less time. Using an efficient data structure helps to scale the crawler when a large number of pages are crawled, making it easy to sort and update the data structure in less time. Storing in memory the links can reduce the speed of crawler if crawling a large number of pages and not using an efficient data structure. As we are crawling nearly ten thousand pages, we will be using an efficient data structure where we can update the promise of the URLs and sort them accordingly.

2.7.1 Priority Dict

A priority queue with updatable priorities You may have needed a priority queue with the ability to change the priorities of elements that were in the queue. This recipe solves this problem in an efficient way. The priority queue is implemented as a dictionary, where keys are the items of the queue, and values are their priorities.

We use Dictionary as priority queue because we want to update the promise as we visit the same link again and using a heap for that will not allow us to update the promise in the heap. So we use the Dictionary as a priority Queue.

Keys of the dictionary are items to be put into the queue, and values are their respective priorities. All dictionary methods work as expected. The advantage over a standard heapq-based priority queue is that priorities of items can be efficiently updated (amortized O(1)) using the code as 'thedict[item] = new_priority.' The 'smallest' method can be used to return the object with the lowest priority, and 'pop_smallest' also removes it. The 'sorted_iter' method provides a destructive sorted iterator.

We are using a heap to modify the dictionary accordingly and update the promise as we crawl further and making sure the time complexity is less. We apply certain methods to keep the dictionary in order and in less time complexity and be able to update the values in the dictionary.

2.7.2 Natural Language ToolKit

NLTK has been called “a wonderful tool for teaching and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing the linguistic structure, and more.
NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

2.7.3 WordNet

WordNet is a part of NLTK which was used to find the synonyms of the given query words in the document. To rank the documents we use Cosine formula in which we calculate the score on the basis of how many times the word occurs in the query. As many documents use a synonym, the score can be improved of the document, if these words are also included. So. including the synonym helped to improve the ranking of the document. Each synonym word is given a score of 0.5 as compared to 1 which is given to the actual word. This is done so that we could also measure the semantic similarity between the query and the document. This helps to improve the document score and get more relevant documents. WordNet is a semantically oriented dictionary of English, similar to a traditional thesaurus but with richer structure. NLTK module includes the English WordNet with 155 287 words and 117 659 synonym sets that are logically related to each other.
Drawback: taking constant 0.5. Would like to equate the occurrence with the similarity score of the query and the synonym.

2.8 Architecture

Below is the system architecture which defines our architecture overall. The steps involved in our focused crawler architecture are

- Get the user query and search it on Google
- Get the top ten results from Google and give each page an initial promise
- Get the link with the highest promise
- Check if it is crawlable or not
- If it is not crawlable, then skip the page
- If crawlable then open the page and get all the links from the page
- Calculate the relevance score of the page
- Calculate promise of all the links and put it in a heap
- Go to the next highest promise page

- Repeat the steps until N pages are crawled
- After crawling N pages, print all the N pages according to their relevance

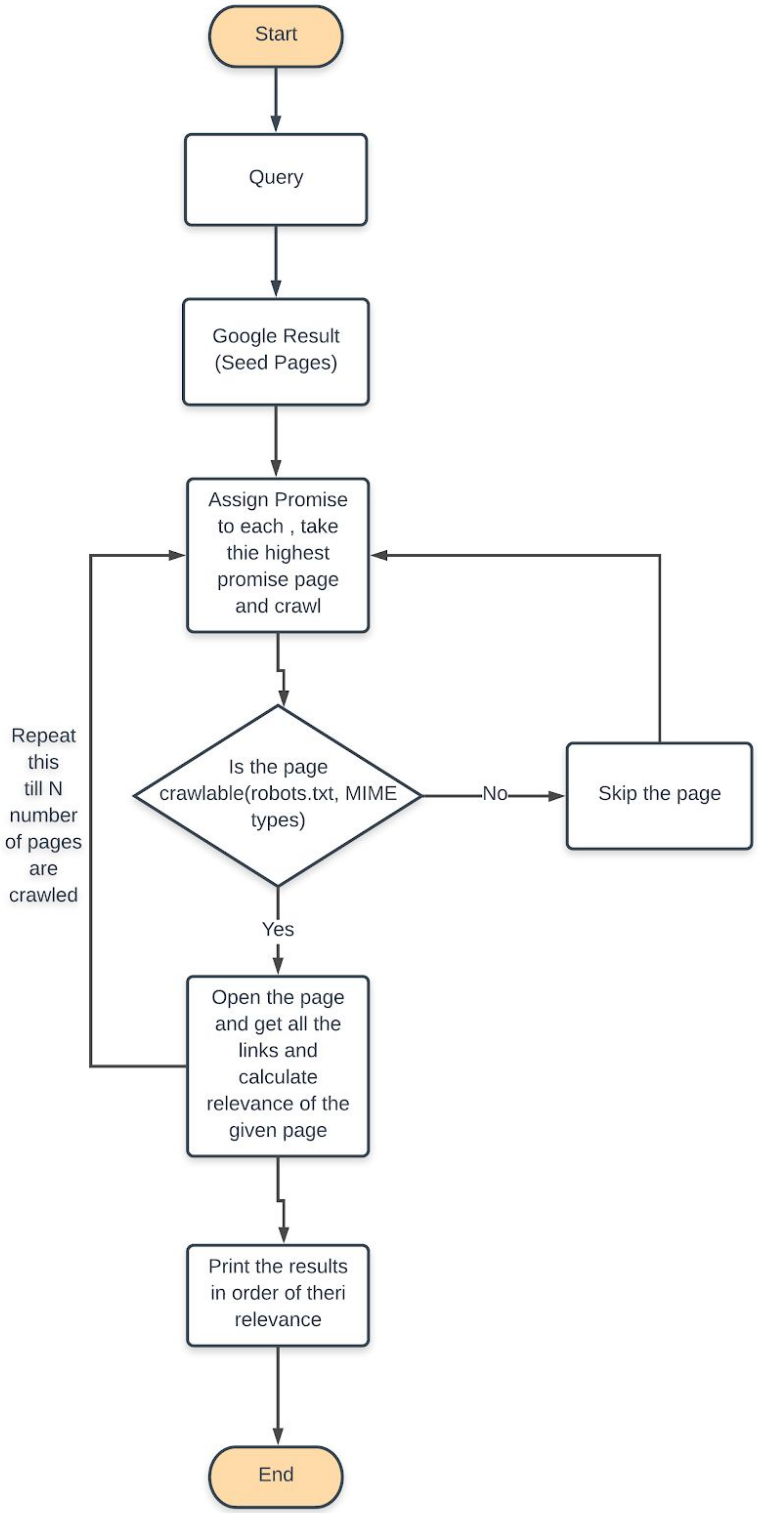

Fig 1: Crawler Architecture

3. Results

To determine the performance of the focused crawler, we look at the Harvest Rate, which is defined as follows:

$$Harvest\ Rate\ =\ \frac{Number\ of\ Relevant\ Pages\ Crawled}{Total\ Pages\ Crawled}$$

Since the cosine measure we use does not have a minimum or a maximum, we choose the threshold as the median of the top 10 cosine measures. If a page has a cosine measure greater than this threshold, we classify the page as 'relevant.'

We ran the crawler on 5000 pages for two types of queries:

1. Large Topic - 'election results'
2. Rare Topic - 'brooklyn dodgers'

	Harvest Rate for Large Topic	**Harvest Rate for Rare Topic**
Crawler without M.L	0.3456	0.0844
Crawler with M.L	0.8272	0.7374

Table 2: Crawlers with their Harvest Rates

The time to crawl 5000 pages was around 3 hours ~ 27 links/min

4. Conclusion

Focused crawler using Machine Learning was designed to improve the quality of pages crawled that are relevant to a user's query. After a deliberate design experiment, we choose Support Vector Regressor as our Machine Learning model for the experiment. We observe that the harvest rate significantly improves when Machine Learning is applied to the crawler.

5. Future Work

In this project, we applied Machine Learning to focused crawling. We modeled the Machine Learning strategy in a query independent way and conducted a scientific experiment to determine the best Machine Learning strategy. Based on this strategy, we were able to improve the Harvest Rate of our focused crawler.

In future work, we hope to conduct the experiment on a larger corpus of documents, over 1 million pages. To achieve this, we will need to make our focused crawler more efficient and scalable. We also aim to investigate other metrics such as the BM25 score instead of the cosine measure to judge the relevant of the page. We would also like to evaluate an online Machine Learning technique such as reinforcement learning to see whether there is any significant improvement in the harvest rate over the offline Machine Learning techniques. Lastly, we would also like to run this on multiple large topic and rare topic queries to evaluate the variance of the harvest rate score.

6. References

- Learning to Crawl: Comparing Classification Schemes: http://homepage.divms.uiowa.edu/~psriniva/Papers/TOIS05.pdf
- A Short Introduction to Learning to Rank: http://times.cs.uiuc.edu/course/598f16/l2r.pdf
- StackOverflow: www.stackoverflow.com
- Medium: www.medium.com
- NLTK Library: www.nltk.org
- Python Libraries: www.docs.python.org/3/library