SQUINT – SVM for Identification of Relevant Sections in Web Pages for Web Search

Siddharth Jonathan J.B.
Department of Computer Science
Stanford University
jonsid@stanford.edu

Riku Inoue
Department of Computer Science
Stanford University
rikui@stanford.edu

Jyotika Prasad
Department of Computer Science
Stanford University
jyotika@stanford.edu

ABSTRACT

We propose SQUINT – an SVM based approach to identify sections (paragraphs) of a Web page that are relevant to a query in Web Search. SQUINT works by generating features from the top most relevant results returned in response to a query from a Web Search Engine, to learn more about the query and its context. It then uses an SVM with a linear kernel to score sections of a Web page based on these features. One application of SQUINT we can think of is some form of highlighting of the sections to indicate which section is most likely to be interesting to the user given his query. If the result page has a lot of (possibly diverse) content sections, this could be very useful to the user in terms of reducing his time to get the information he needs. Another advantage of this scheme as compared to simple search term highlighting is that, it would even score sections which do not mention the key word at all. We also think SQUINT could be used to generate better summaries for queries in Web Search. One can also envision SQUINT as being able to create succinct summaries of pages of results, by pulling out the most relevant section in each page and creating a meta summary page of the results. The training set for SQUINT is generated by querying a Web Search Engine and hand labelling sections. Preliminary evaluations of SQUINT by K-fold cross validation appear promising. We also analyzed the effect of feature dimensionality reduction on performance. We conclude with some insights into the problem and possible directions for future research.

Keywords

Artificial Intelligence, Machine Learning, Supervised Learning, SQUINT, Information Retrieval, Web Search, Support Vector Machine.

1. INTRODUCTION

Currently for Web Search, the de-facto model is that a user inputs a query and gets multiple pages of links to results. Then the user, based on the snippets that he sees, clicks on one of the results. The result page could have a lot of content in it and it is left to the user to figure out where in the page, his answer to the query lies. Some shallow techniques that search engines at times use to offset this problem are to highlight the search terms in the document. Although this is a step in the right direction, the results are often very unsatisfactory especially if the search term is not a very differentiating term or if it does not appear in the section at all.

In this paper, we propose SQUINT, an SVM based approach for identifying the sections in a web page relevant to a user's query to help the user jump right into the most relevant section of the Web page. Here, we define a section to be a paragraph. SQUINT works by making use of the top K results returned in response to a query from a Web Search Engine (We use the Google API), to learn more about the query and its context.

The problem can be formalized as follows. Given a query Q, a standard Web search engine returns a set of pages ranked in descending order of relevance. We call this set, P. Given P, let S be the set of all sections in every page in P.

$$\mathbf{S} = \{s;\ s \text{ is a section in some page in P}\}$$

Given Q, we generate S, and use it to train a Support Vector Machine. During testing, given a particular section, we can predict a numerical score indicative of the relevance of the section to the query, Q. The scores of all the sections in a page with respect to a query enforce an ordering among the sections in terms of relevance to the query. We use an SVM with a linear kernel in SQUINT. Support Vector Machines have been shown to work very well for high dimensional learning problems similar to the ones encountered when dealing with text [8].

The SVM is trained with binary class labels for relevant and irrelevant classes and we use the SVM's predicted margins during testing for scoring sections.

In some sense, we identify related concept words to the query based on term frequency and proximity statistics. We then build query independent features based on occurrence of these high frequency words of the top K results, in the section to be classified.

We begin by discussing some related research in Section 2. In Section 3, we first present a high level overview of the SQUINT framework and then subsequently drill down into the individual sub components in the framework. In section 4, we report results obtained during the training and test phases for various experimental settings. In section 5, we present some insights into the problem that we gained during the course of our experimentation. We end with section 6, where we present our conclusions and possible future directions for research.

2. Related Work

Gelbukh et al. [1] have presented a model for recognition of relevant passages in text, using relevance measures and structural integrity. Liu and Croft [2] have explored passage retrieval using a language and relevance model. This problem finds applications in web-question answering and summary generation, and has been addressed in these contexts in [3], [4], [5]. Yu et al. [6] discuss a Vision-based Page Segmentation (VIPS) algorithm to detect the semantic content structure in the page, akin to identifying the sections. Teevan et al. [7] have discussed search methodologies that focus more on contextual information than just keyword occurrences.

3. Overall Architecture

As shown in Fig.1 below, the training phase begins with querying Google and receiving the ranked set of relevant pages. These pages are then cleaned and split by the *Page Processor* module, which outputs a set of sections. The *Feature Generator* then computes the features for each section. Every section is then manually labelled in the labelling phase. We use a binary labelling scheme where the labels are +1(relevant) and -1(irrelevant). The output of these two phases gives us the training set, which is used to train the SVM.

After learning is complete, we proceed to the testing phase where, given a query and a set of result pages, we are able to score the sections in the result pages based on the features we extract from the result set. The SVM's predicted margins are used as scores here.

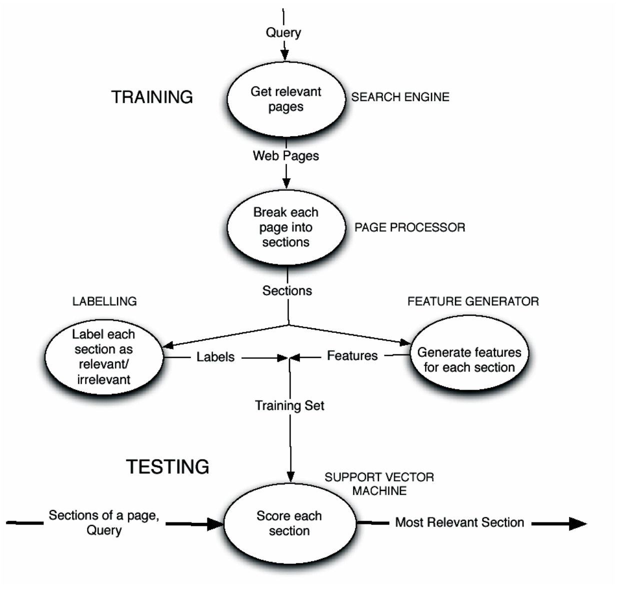

Figure 1: SQUINT – High level view of the framework

3.1. Feature Generation

In order to effectively score sections of a web page using the SVM, we need to select features that capture each section's characteristics. Among the information extracted from sections, the frequency of certain 'important' words and the location and frequency of the query words are considered to be useful indicators of relevance to the queried topic. The intuition here is that certain words which are strongly related to the topic will occur frequently in relevant sections, and also relevant sections are located near the query words. In the proposed method, we currently have five possible types of features that capture word frequency and word location.

3.1.1. Word Rank Based Features

We define the rank of a word to be its position in the list if the words were ordered by frequency of occurrence in the top K results. We would have a feature each for say, the top 300 most frequent words in the top K results. For the i th ranked word, this feature would basically have the value for the frequency of this word in the current section. One possible option that we have to limit the dimensionality of the input vector is by bucketing words by a certain range of ranks. For example, we can bucket ranks 1-5, 6-10, 11-15...etc to aggregate word counts, and come up with a feature vector of reduced dimensionality. Another option for limiting the dimensionality of the input vector is to simply limit the range of ranks that appear in the vector. Figure 2 shows the effect of dimensionality reduction on the accuracy of the test result. Bucket size here is 1 since it worked the best among various bucket sizes tried. After testing various settings for bucketing and the range of ranks, we decided to use a bucket size of 1 and rank coverage of 150 which achieved the highest accuracy. In other words, we use top 150 ranking words with no bucketing. We also normalize for the length of the section since we do not want to be biased towards long sections.

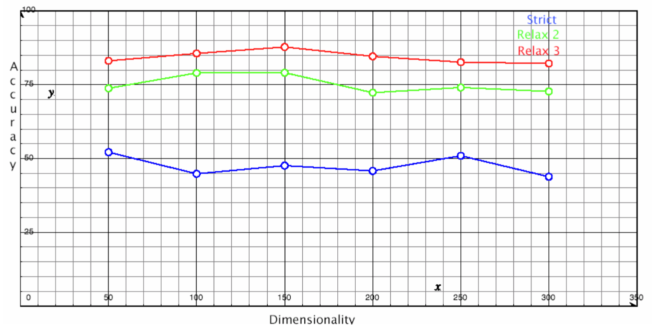

Figure 2: Dimensionality Reduction in Word Rank Features

3.1.2. Bigram Rank Based Features

We define a bigram to be two consecutive words occurring in a section. This feature is computed in a manner similar to the previous set of features. This feature is based on the intuition that the correlation between two words might be more informative than the words taken individually. For instance, "machine learning" suggests a stronger relation to a query "AI SVM" than the individual words "machine" or "learning". For this feature as well, we adjust the dimensionality by bucketing and limiting coverage of ranks. Figure 3 shows the effect of dimensionality reduction on accuracy. A bucket size of 1 worked best for this feature as well. From these results, we decided to use a bucket size 1 and rank coverage 50.

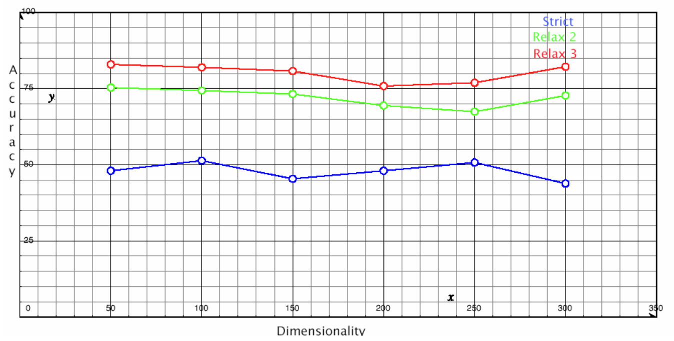

Figure 3: Dimensionality Reduction in Bigram Rank Features

3.1.3. Coverage of Top Ranked Tokens

Relevance to a topic may also be captured by the coverage of top ranked token types in the section. For example, if we have a bucket size of 5, we might be interested in knowing how many of the top 5 ranked words occur in this section, how many of the next 5 highly ranked words occur in this section and so forth. Specifically, if the top 5 ranked token types are "learning", "machine", "data", "access", and "database", and a section contained "learning" and "data", the corresponding value for this feature is 2. We use bucket size of 5 and dimensionality of 30 for this feature.

3.1.4. Distance from the Query

The intuition here is that the closer a section is to the query in the Web page, the more likely it is to be relevant. Thus we compute the section-wise distance between the section in question and the nearest section which contains the query. We feel that although this is not a necessary condition for relevance, it could well be a sufficient one. An ablative analysis with respect to this feature confirmed this intuition.

3.1.5. Query Word Frequency

Last but not the least, the frequency of the query word in the section seems a reasonable indicator for relevance. Since there are many possible ways to evaluate importance of query word appearance, we tried two types of measurements for this feature. First, we used the query term frequency in the section. In this setting, if there are 3 query words that appear in a section, the count is 3 normalized by the number of words in the section. Second, we used a sum of weighted counts based on the distance from the beginning of the section. Research from text summarization has shown that typically the gist of a paragraph is given by its first few sentences. In other words, a match in the first few words of a section counts more than a match much lower down in the section. Weighted count is computed by the following equation.

$$weightedcount = \frac{(sectionsize - position)}{sectionsize}$$

In other words, we discount the count linearly as a query word occurs in the latter part of the section. For instance, if a query word occur as the 20th word, and the total number of words in the section is 100, the weighted count is 0.8. After testing both settings, we decided to use the weighted count setting.

3.1.6. The Final Set of Features

The features discussed thus far are generated for each section in the top K result pages obtained by querying Google. We put the features together and evaluated each setting by comparing K-Fold Cross validation accuracy to decide the optimal combination of the features. After feature selection, we decided to use the set of features shown in table 1. We will explain the details of feature selection in 3.4.3.

Table 1: The Final Set of Features

Feature Name	**Parameters**
Rank Based	Dimensionality: 150 No Bucketing
Bigram Frequency Based	Dimensionality: 50 No Bucketing
Coverage of Tokens	Dimensionality: 30
Distance from the Query	Dimensionality: 1

3.2. Training Set Generation

The training set required is a set of sections of web pages and corresponding binary labels indicating +1(relevant) and -1(irrelevant). We created the training set by hand labelling the sections of pages returned by Google on a few sample queries. The basic steps are,

1. Query Google to get a set of pages
2. Clean each page – remove scripts, pictures, links etc.
3. Break each page into sections.
4. Label each section of every page.

Step 1 uses the GoogleSoapSearchAPI. A quick way to do step 2 is to get a Lynx dump of the web page. Lynx being a text based browser cleans up scripts and pictures, and gives text with numbered parts, where each part is a distinct html element. We use this for Step 3. The page is broken up into candidate sections based on the numbering. Candidates which have less than 2 lines of text are eliminated, as we are only interested in significant chunks of text. Lynx also groups all the links on the page under 'Visible Links' or 'References', both of which are removed.

One caveat here is that we need to distinguish between training labels and test labels. For training, we hand labelled every section as +1 (relevant) or -1 (irrelevant). During testing, our task is actually to detect on a per page basis, the most relevant section in that page. Therefore our labelling is slightly different. For every page, the most relevant section(s) is(are) labelled as 1, while all others are labelled as 0. Note that we label the test set for the purposes of evaluation only.

We generated the data set from 6 queries – *"machine learning"*, *"gene sequencing"*, *"oregon missing family"*, *"space shuttle discovery launch"*, *"ipod nano"* and *"google buys youtube"*.

3.3. Learning Algorithm

As mentioned earlier, we use a Support Vector Machine with a linear kernel to learn to detect the most relevant section in a given page, using the training set mentioned in the preceding section. The training set contains results for 6 queries which comprises of 94 web pages and 775 sections. Given the relatively high dimensionality of our feature vector, it is a reasonable choice to use an SVM. Note that our purpose is to specify the most relevant section, not just classify many relevant sections. To get a non-binary metric of how relevant sections are, we use the predicted margins for each sample. In other words, given parameter w and feature vector x, we detect a sample that has the largest $w^T x$ as the most relevant section in the page. Also note that the way the learning algorithm deals with data is different between the training phase and the test phase. In the training phase, all the result pages for a query are processed all at once, but in the test phase, the algorithm examines the data, page by page, to determine the most relevant sections for each page.

4. Evaluation

We evaluate the performance of the learning algorithm using four common metrics namely, K-fold Cross validation, learning curve with respect to number of training data, ablative analysis for features, manual error analysis. In each case, we use three different settings for the evaluation – strict, relax 2 and relax 3.

These three settings can best be explained with an example. Let s1, s2 and s3 be the top three highest scoring sections in a particular page p, as returned by the SVM.

$$score(s1) > score(s2) > score(s3)$$

Under the strict setting, a result is deemed correct only if s1 is the most relevant section in the page, as indicated from the labelling in the test data. Under relax 2, the result is deemed correct if either s1 or s2 is the most relevant section in the page. Similarly, for relax 3, the result is correct if either s1 or s2 or s3 is the most relevant section in the page.

4.1 K-fold Cross validation

We do k-fold cross validation with k = 6. The 6 datasets were the results for the 6 queries. We evaluate the accuracy for the strict, relax 2 and relax 3 settings. In each case, we first evaluate the accuracy per query as the percentage of pages within the query for which the SVM returned a correct result. The k-fold accuracy is then the average of the accuracies obtained for all the 6 queries. The following results were obtained.

Table 2: K-fold Accuracy for the Best Configuration

Best Configuration	k fold accuracy
Strict	60.60%
relax 2	80.95%
relax 3	90.47%

4.2 Learning curve

We examine how much training data we need, to get reasonable accuracy by plotting the learning curve with respect to the number of training examples. The horizontal axis is the size of training set, and the vertical axis is the accuracy as measured by k-fold cross validation.

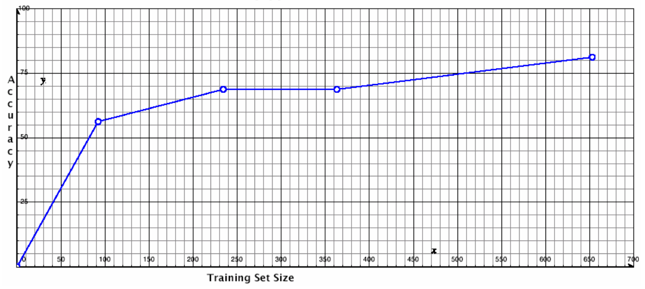

Figure 4: Learning Curve

4.3 Forward feature search

We also measured the significance of each feature through forward search on the feature set. We started with the base case (lower bound), where the algorithm randomly picks a section as the most relevant. We added the frequency-based features (word frequency and bigram frequency) next, followed by coverage and distance from the query. The last feature added was the frequency of the query word in the section. We were able to eliminate the section size feature, since we observed that we did not get any gain in accuracy by the use of this feature.

The following chart shows the results of the forward search. 1 indicates the frequency based features, 2 indicates the coverage of top ranking words, 3 is the distance of the section from the query and 4 is the frequency of the query terms in the section.

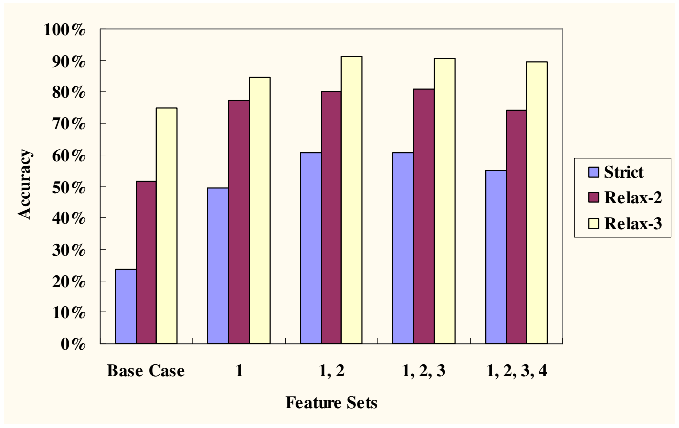

Figure 5: Forward Feature Search

An unexpected observation was that the use of the query frequency feature actually marginally hurt the accuracy of prediction under the strict and relax 2 settings. We hypothesize that the lack of a gain in accuracy with this feature could be because this information is captured somewhat noisily by the rank frequency and coverage features.

5. Insights

There might appear to be a potential problem for SQUINT in the case of queries which result in unequally sized clusters in the result set. For example, for the query "Michael Jordan" one might expect a majority of the results returned to talk about the basket ball player. However, we would like to identify sections mentioning the Professor from Berkeley as also relevant. Given the current SQUINT framework it seems reasonable to expect that it will be highly unlikely that relevant sections belonging to the minority cluster will be correctly identified. However, we claim that this problem is orthogonal to the one SQUINT attempts to solve and the preceding scenario can be easily resolved by allowing SQUINT to operate on a per cluster basis.

Another observation we made was that the frequency based features were not as useful as we had hoped. In fact, the word coverage feature is more critical to the accuracy of the scoring. We think that this might be explained by the fact that since we do not do idf weighting, our frequency features are susceptible to noise resulting from low-idf words. We do stop word filtering but that may not be enough. The coverage feature is a little less susceptible to this effect since it is bucketed and on inspection of the most frequent words more often than not the top ranking words are words that are highly correlated with the query.

Given the limited size of our training set, we attempted to reduce the dimensionality of some of our features. As shown in an earlier section, we observed that reducing the dimensionality did give us gains in accuracy, presumably because of the reduced number of parameters that need to be fit for the training set.

We realize that there are quite a few reasonable extensions to our feature set. We intend to explore some of these in the future. One obvious upgrade to our suite of frequency features is to weight it by idf. In addition, the query frequency features can be encoded in a number of ways. For example, we could have a feature that looks for all the words in the query to occur within a specified window of words and counts occurrences only when the query words occur within that window. One can imagine that this might be useful for a query like “data mining”, wherein if the words “data” and “mining” occur far apart, the meaning conveyed is not quite the same as the query. One can also imagine designing features that penalize absence of any of the query words in the section. Many of these features are similar to the query based features that a Web Search Engine or any information retrieval system might employ.

Another interesting observation that we made was that the relevance of a section did not seem to be too correlated with the length of the section. The use of the length feature did not hurt accuracy but it did not give us noticeable gains either.

We realize that SQUINT offers a value add for a specific category of queries in Web Search and information retrieval namely, 'information seeking' queries. In such queries, the focus of the query is reasonably broad and good result pages comprise many sections of text. For example, *'gene sequencing'*. However, for say commercial queries, where a good result page is a home page of a dealer or a hub page with lots of outgoing links, graphics and animation, the value add is questionable.

5. Conclusion and Future work

We proposed SQUINT, an SVM based approach to identify relevant sections in a Web page to a user's search query. This problem has been relatively less studied in the literature, but we believe that its solution will have a large impact on the user's overall search experience. As a result of our evaluation, we see that using information retrieval inspired features and some basic hints from summarization give respectable accuracy with respect to detecting the most relevant section in a page. Some possible future directions include qualitative comparisons of the summaries generated using SQUINT with the snippets generated by Web Search Engines and other summarization algorithms. It will also be interesting to evaluate the impact SQUINT has on user productivity with regard to satisfying user information need.

Acknowledgements:

The authors would like to thank Prof. Ng, Jeremy Kolter, Samuel Ieong, Catie Chang, Haidong Wang and Eric Delage for their support and feedback throughout the course of the project. We also wish to thank Dr.Andreas Paepcke of the Stanford InfoLab for his insightful comments and suggestions.

REFERENCES

[1] Alexander Gelbukh, NamO Kang, and SangYong Han. Combining Sources of Evidence for Recognition of Relevant Passages in Texts. *ISSADS 2005*, pp. 283–290, 2005

[2] Xiaoyong Liu, W. Bruce Croft. Passage retrieval based on language models. In *Proceedings of the eleventh international conference on Information and knowledge management*. pp. 375 – 382, 2002.

[3] Jimmy Lin, Aaron Fernandes, Boris Katz, Gregory Marton, Stefanie Tellex. Extracting Answers from the Web Using Knowledge Annotation and Knowledge Mining Techniques. In *Proceedings of the Eleventh Text REtrieval Conference (TREC 2002)*, Gaithersburg, Maryland, 2002.

[4] Brian Ulicny. Lycos Retriever: An Information Fusion Engine. *In Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, pp. 177–180, June 2006.

[5] Hang Cui, Renxu Sun, Keya Li, Min-Yen Kan, Tat-Seng Chua. Question answering passage retrieval using dependency relations. *In Proceedings of the 28th annual international ACM SIGIR conference on Research and development*. pp. 400 – 407, 2005.

[6] Shipeng Yu, Deng Cai, Ji-Rong Wen, Wei-Ying Ma. Improving Pseudo-Relevance Feedback in Web Information Retrieval Using Web Page Segmentation. *In Proceedings of the 12th international conference on World Wide Web.* pp. 11-18, May 2003.

[7] Jaime Teevan, Christine Alvarado, Mark S. Ackerman and David R. Karger. The Perfect Search Engine Is Not Enough: A Study of Orienteering Behavior in Directed Search. *In Proceedings of the SIGCHI conference on Human factors in computing systems.* pp. 415-422, April 2004.

[8] Thorsten Joachims, Text Categorization with Support Vector Machines – Learning with many relevant features. *In Proceedings of the 10th European Conference on* Machine Learning pp. 137-142, 1998.

[9] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsv