Tibianna: A Learning-Based Search Engine with Query Refinement

Clint Heyer *Joachim Diederich*
Information Environments Program
School of Information Technology and Electrical Engineering
The University of Queensland,
Brisbane, 4072 Australia
clint@thestaticvoid.net *joachimd@itee.uq.edu.au*

Abstract

While web search engine technology has improved over time, there is often a fundamental reliance on keyword matching for searches. What happens however, when the user does not know what keywords to use?

This paper presents preliminary learning results of a prototype learning search engine that attempts to address this problem. Tibianna allows a user to manually rank a set of results based on their own relevancy function. Once a required number of results are ranked, the set is downloaded, processed and presented to support vector machines (SVMs) for learning. Once learned, Tibianna can actively reorder or discard search engine results based on the model it has learned. This provides a way of improving search results without requiring query refinement. Learning outcomes from experimental trials with Tibianna are presented, demonstrating the implications of using different preprocessing techniques and corpus sizes.

Query refinement functions are also available to the user, which can enable exploration of query words via the WordNet database, and allows quick query refinement via a dynamic HTML interface.

Keywords Information Retrieval, Personalised Documents, Search Engine Technology

1 Introduction

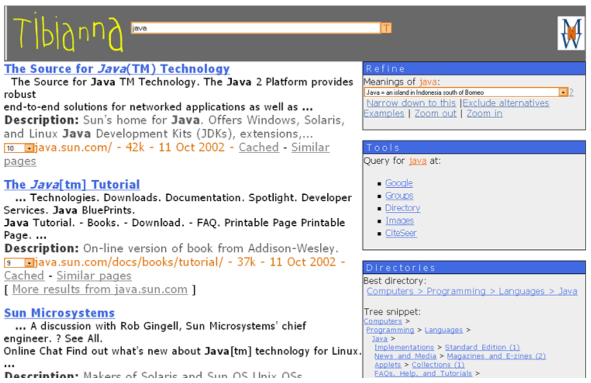
This paper attempts to address an inherent problem with keyword-based search engines – what happens if the user doesn't know what keywords to use? This situation can present itself when the user is searching new domains; when the user is uncertain about what to search for or when they do not know what keywords to use to get the desired results. There is a divide between the user's search *intent* – their mind view of what kind of results they wish to find - and the effectiveness of the query they craft to match their intent. How effectively a user can craft their query is largely dependant on their knowledge of the search domain and proficiency with the search engine used.

The prototype implementation, named *Tibianna*, presents the user with query refinement options by using contextual and semantic data about the user's query, gathered from WordNet[1]. Supervised machine learning is used by the system in order to provide the user with results that more closely match their search intent without necessitating modification of the search query. Learning what the user's information needs are is a powerful technique; and has the possibility of increasing the effectiveness of a search engine. This latter method will be the primary focus for this paper, and indeed the system itself.

By providing the user with more tools for refining their search, the system can better capture what the user values. This value metric varies from query to query and user to user, even though keywords used are identical[1, 2].

Support vector machines (SVMs)[3] have been shown to be highly effective in text classification problems, primarily due to their ability to handle large dimensionality and a large number of features[4].

Tibianna takes the form of a traditional search engine, with interface and result-formatting similar to Google. The user interacts via a web browser, with Tibianna residing on either a remote server or the local machine. In preliminary experiments, Tibianna utilised Google as its underlying provider of search results. After the user enters a query, each result is displayed with a drop-down box allowing the user to rank its relevancy, after either viewing the page or by reading the page summary.



The Source for *Java*(TM) Technology
The Source for **Java** TM Technology. The **Java** 2 Platform provides robust
end-to-end solutions for networked applications as well as ...
Description: Sun's home for **Java**. Offers Windows, Solaris, and Linux **Java** Development Kits (JDKs), extensions,...
10 java.sun.com/ - 42k - 11 Oct 2002 - Cached - Similar pages

The *Java*[tm] Tutorial
... Technologies. Downloads. Documentation. Spotlight. Developer Services. **Java** BluePrints.
Java Tutorial. - Books. - Download. - FAQ. Printable Page Printable Page. ...
Description: On-line version of book from Addison-Wesley.
9 java.sun.com/docs/books/tutorial/ - 37k - 11 Oct 2002 - Cached - Similar pages
[More results from java.sun.com]

Sun Microsystems
... A discussion with Rob Gingell, Sun Microsystems' chief engineer. ? See All.
Online Chat Find out what's new about **Java**[tm] technology for Linux.
...
Description: Makers of Solaris and Sun OS Unix OSs

Refine
Meanings of java:
Java = an island in Indonesia south of Borneo ?
Narrow down to this | Exclude alternatives
Examples | Zoom out | Zoom in

Tools
Query for java at:
- Google
- Groups
- Directory
- Images
- CiteSeer

Directories
Best directory:
Computers > Programming > Languages > Java

Tree snippet:
Computers >
Programming > Languages >
Java >
Implementations > Standard Edition (1)
News and Media > Magazines and E-zines (2)
Applets > Collections (1)
FAQs, Help, and Tutorials >

Figure 1: Initial search result page

After ranking a number of results the user can opt to send the rankings back to Tibianna. Tibianna then downloads each result and processes it, displaying the next page of results to the user. Once a requisite number of results are ranked by the user, Tibianna switches to classification mode, and reorders Google results according to the model it has learned.

Section 2 examines how features are extracted and processed from web documents in Tibianna, with Section 3 introducing the query refinement options briefly, and showing screen shots of the features. Section 4 details parameter experiments, with the results presented in Section 5. Section 6 discusses

[1] http://www.cogsci.princeton.edu/~wn/

**Proceedings of the 7th Australasian Document Computing Symposium,
Sydney, Australia, December 16, 2002.**

future work with conclusions drawn in Section 7. More background on Tibianna is presented in [5].

2 Feature Extraction

Resources in Tibianna undergo a series of transformations before they can be presented to the SVM for learning. Firstly, HTML mark-up and any other special symbols that do not form part of the content are removed. The obvious side-effect of this stage is that any directly or indirectly encoded metadata is discarded, and cannot influence the learning result. Other work such as [2, 6] maintains this data to add weighting to subsections of content.

In the next stage, a stop list is used to remove the most 300 commonly used words in the English language. Words remaining from this stage are then stemmed according to the Porter algorithm[7]. Stop word removal takes out a number of features that have little bearing on the meaning of a resource and stemming reduces the number of features by converting plural and tense variations of a word to a common stem. N-grams (up to 5-grams are used in Tibianna) are then created based on remaining words.

Finally, features extracted from the entire corpus are brought together to be counted and weighed. Inverse document frequency (IDF) weighing is used to reduce the weight of commonly occurring features. The output of this stage is a training file used by SVM^{light} [8] to create a model from the data. This model is later used in the classification stage, after resources have gone through the same pre-processing stages as the training resources. Tibianna uses SVM ranking classification[6], as opposed to the more traditional binary classification.

3 Query Refinement

To assist the user in refining their search, query refinement options are available. These options are of particular value when the user is searching in a new domain, and makes use of the lexical database, WordNet. WordNet has been used in other work for query refinement, such as [9, 10].

Search queries are split into individual words and cross-referenced with WordNet. Data such as senses, synonyms, hyponyms, holonyms and meronyms for each word is presented to the user in a dynamic HTML interface within the search results. Client-side scripting is used for the interface to allow the user to explore the refinement options quickly, without the extra roundtrip to the server. Usage examples of query refinement using the query "*java*" are shown in Figures 2-4.

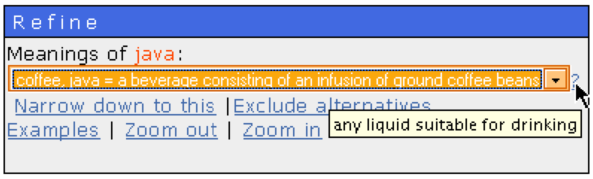
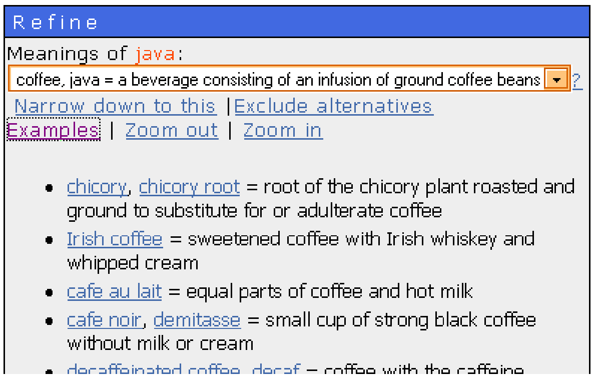

Refine
Meanings of java:
coffee, java = a beverage consisting of an infusion of ground coffee beans ?
Narrow down to this | Exclude alternatives
Examples | Zoom out | Zoom in
any liquid suitable for drinking

Figure 2: Each query word has a 'Refine' box with lets the user explore the word, and modify their query.

The '?' text provides a tooltip for the meaning of a sense.


Refine
Meanings of java:
coffee, java = a beverage consisting of an infusion of ground coffee beans ?
Narrow down to this | Exclude alternatives
Examples | Zoom out | Zoom in

- chicory, chicory root = root of the chicory plant roasted and ground to substitute for or adulterate coffee
- Irish coffee = sweetened coffee with Irish whiskey and whipped cream
- cafe au lait = equal parts of coffee and hot milk
- cafe noir, demitasse = small cup of strong black coffee without milk or cream
- decaffeinated coffee, decaf = coffee with the caffeine

Figure 3: A user can see a list of examples of a sense meaning. Each example is hyperlinked and adds the example to the query when clicked.

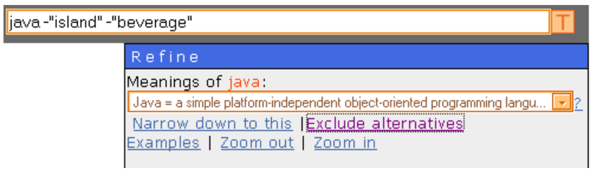

java -"island" -"beverage"
Refine
Meanings of java:
Java = a simple platform-independent object-oriented programming langu... ?
Narrow down to this | Exclude alternatives
Examples | Zoom out | Zoom in

Figure 4: Clicking 'Exclude alternatives' attempts to reduce result set size by explicitly excluding alternative senses. In this example, the original *"java code"* query is refined to *"java code –'island' –'beverage'"*

4 Experimental Methodology

Many of the pre-processing stages have parameters that need to be determined for optimal running of the system. To discover these parameters, various experiments were run with real-world data sets. The training document set of 28 pages (see discussion below on the size of the training set) were sourced from the first 30 results for the Google search of *'java'*. Each result was hand-ranked on a 0 to 10 scale, with 10 being the highest. Another set of 11 pages were sought, coming from different sites than those used for training data. Some of these pages were on-topic, others completely irrelevant. These test pages were again hand-ranked using the same scale as the training pages so the classification results could be compared against a consistent benchmark. The rank frequency of the training documents is shown in Table 1.

Rank	*Frequency*
0	5
1	7
2	7
3	2
4	1
5	1
6	1
7	1
8	0
9	0
10	3
	28

Table 1: Rank frequency distribution for training documents

Note the unbalanced frequency as it makes learning more difficult for the SVM, but in turn is a more realistic test case: the search engine does not return many relevant results - exactly the problem Tibianna is designed to address.

To determine the effectiveness of the classification results consistently, a simple scoring algorithm was created. The score of a ranking classifier output is more difficult to determine than the traditional binary class based classifier output because a ranking is more complex, being multidimensional. The algorithm maximises the score for orderings that most closely match the target ordering. Relevant attributes are: (1) ranks that are in exact sequence (even though possibly incorrectly ranked), (2) distances between actual and target ranks, and (3) the number of ranks which are "roughly" in sequence (see below). An exact sequence is defined as an ordering such as *2, 3, 4, 5* in which ranks are in exact order, but may be offset from the correct rank. A "rough" sequence is defined as a series of ranks which maintain the correct ordering relationships, but may have one or more missing ranks, for example: *1, 3, 4, 6*.

A comparison between scored output and leave-one-out (LOO) SVM error rates may serve to highlight the differences, where the target ordering is *0...10*:

Ordering produced	*LOO Error*	*Score*
1 6 3 2 4 8 0 5 9 7 10	5.59%	18
1 2 3 6 4 8 5 0 7 9 10	6.83%	31
1 2 3 6 4 8 5 0 7 9 10	10.56%	31

Table 2: An illustration of the differences in leave-one-out (LOO) error and score.

As shown in Table 2, the score is determined by how well the SVM ranked a test set, and may or may not be indicative of a good learning result.

5 Experimentation

5.1 The effect of pre-processing options and transformations

In this trial, a static training and test document set was used, with feature extraction parameters systematically modified. There are several interesting conclusions that can be drawn from this trial, which are graphed in Figure 5.

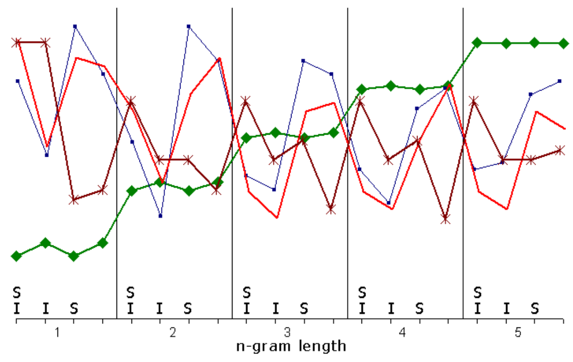


Figure 5: Parameter tuning results. The major horizontal axis is the n-gram length. The minor horizontal axis represents whether stemming and IDF weighing was used; just IDF was used; just stemming was used 'S'; or neither, denoted 'SI', 'I', 'S' and '' respectively.

Most importantly, it is apparent that the learning error rate (leave-one-out error rate, indicated by unmarked line) drops significantly when resources are pre-processed using inverse document frequency (IDF) to weigh features. Using both IDF weighing and word stemming was not as efficient however.

The number of support vectors (SVs, shown with a square-marked line) required for learning closely tracks the error rate. This is an indication that as a result of a difficult to learn problem, the number of SVs required to represent the problem grows. As the n-gram count grows (and with it, feature counts – denoted in diamond-marked line), the maximum bound for the number of SVs and the error rate appears to decrease; although the lower bounds remain relatively stable.

If we look at the score (defined in section 4, shown as a star-marked line) which has been calculated for each test, we can see that that it has a roughly inverse relationship to the error rate. This lends weight to the accuracy of the scoring algorithm, as a high error should result in a low score. Interestingly, when the error rate is in troughs (in the IDF processing column for each n-gram group), the score drops to a median level when it should be highest. Over the period of 2-5 grams, a clear pattern emerges, whereby the score is highest when both stemming and IDF-based weighing are utilised, and drops to its lowest when neither are used. While the latter point is consistent with the error rate, the first is not.

5.2 Corpus Size

Exactly how many documents Tibianna requires the user to rank before it can accurately begin to classify results was an open question at the beginning of this work. A balance needs to be sought between how much time and effort is expected of the user, and how many documents are required for classification. Figure 6 plots the results of this trial.

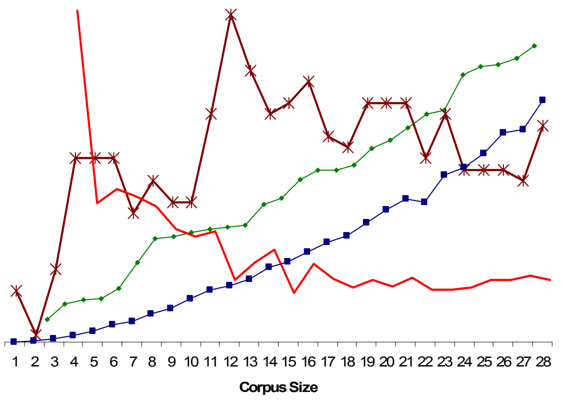


Figure 6: Corpus size trial results

With the rising number of documents (horizontal axis) feature counts and support vector quantities (diamond-marked and square-marked lines respectively) also rise, as expected. The leave-one-out error rate (unmarked line) appears to follow a decay function, dropping very rapidly when the corpus size is low, and remaining stable once the corpus size exceeds 20.

From repeated runs of this trial (with random orderings each time, and with different datasets), it is evident that the error rate does not decrease significantly past 15 documents, and in practise could be set as low as 12. While training on a larger document set can often improve accuracy of learning, the cost is increased processing time and more time and effort required of the user. In general usage by the authors, it was found that training on 15 documents was acceptable in both learning result and user effort terms. The perceived accuracy of the learning result does depend on the number of positive training cases, and the use of a consistent ranking method by the user.

6 Future Work

Future analysis and experimentation could improve the final system. How stop-word list size and contents (e.g. using most common words on web rather than in general language usage) affects learning is something not investigated in this paper, but would be beneficial. While the corpus size was intentionally kept small to simulate real-world usage, an investigation into how Tibianna scales using larger corpus sizes would be beneficial. Other work to be done is to investigate the scoring algorithm more exhaustively, experiment with feature reduction methods, and examine how the frequency of differently ranked results alters learning.

7 Conclusion

Tibianna is a unique system which addresses shortcomings in traditional keyword-based search engines. Primarily, how the user is to find what they desire without knowing the best keywords to use, and how the divide between query terms and the user's search intent be minimised. Tibianna offers users the ability to rank search results based on their own relevancy function. After the user has trained the system on a minimum number of results, Tibianna learns the user's relevancy function with a high degree of accuracy. This learned function is then applied to subsequent search results by reordering results according to learned model.

Assisted query refinement via a client-side HTML interface is used in Tibianna to improve the precision of returned results. By selecting terms from the interface, the user can dynamically refine their search by adding terms to the query.

In general usage by the authors, Tibianna has shown to be highly valuable. With the combined SVM learning and query refinement, Tibianna is an effective system for improving existing search engines.

8 References

[1] E. Glover, S. Lawrence, G. Michael, W. Birmingham, and C. L. Giles, "Web Search - Your Way," *Communications of the ACM*, 1999.

[2] E. Glover, G. Flake, S. Lawrence, W. Birmingham, A. Kruger, C. L. Giles, and D. Pennock, "Improving Category Specific Web Search by Learning Query Modifications," presented at Symposium on Applications and the Internet, SAINT, San Diego, CA, 2001.

[3] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20(3), pp. 273-297, 1995.

[4] T. Joachims, "Text Categorization with Support Vector Machines: Learning With Many Relevant Features," presented at Proceedings of ECML-98, 10th European Conference on Machine Learning, Heidelberg, Germany, 1998, pp. 137-142.

[5] C. Heyer, K. Hollingsworth, J. Madden, P. Heydon, K. Bartlett, and J. Diederich, "MyNewsWave: User-centered Web search and news delivery," (to appear) 7th Australasian Document Computing Symposium, Sydney, Australia, 2002.

[6] T. Joachims, "Optimizing Search Engines Using Clickthrough Data," presented at ACM Conference on Knowledge Discovery and Data Mining, 2002.

[7] M. Porter, "An Algorithm for Suffix Stripping," in *Readings in Information Retrieval*, vol. 14. San Francisco: Morgan Kaufmann, 1997, pp. 130-137.

[8] T. Joachims, *Making Large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning*: MIT-Press, 1999.

[9] P. D. Bruza, R. McArthur, and S. Dennis, "Interactive Internet Search: Keyword, Directory and Query Reformulation Mechanisms Compared," presented at 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2000.

[10] D. Moldovan and R. Mihalcea, "Using WordNet and Lexical Operators to Improve Internet Searches," *IEEE Internet Computing*, vol. 4(1), pp. 34-43, 2000.