Semantic Search in Emails

Navneet Kapur, Mustafa Safdari, Rahul Sharma

December 10, 2010

Abstract

Web search technology is abound with techniques to tap into the semantics of information. For email search, such techniques are almost non-existent in the mainstream. Major email services provide only syntactic searching to their clients. Our idea is to use PCA-based techniques to mine the underlying semantic structure present in the mailbox of a person. For the dataset, we use the email dataset provided by Enron corporation. We demonstrate the applicability of LSI to email search and discuss the various information retrieval techniques used to improve the search performance. Lastly, we propose a method to update the weights so that our search engine learns user preferences over time.

1 Introduction

Email users frequently have problems searching and locating specific emails in their accounts. The primary reason is the email providers only provide syntactic search utilities. There are several problems with syntactic search:

- **Missing results** : If the user query does not contain the exact words which occur directly in the emails then syntactic matching will miss it. Thus this method performs worse if the user gives more information! Since the user does not want to miss the document he is searching for, his queries will be short. This results in the next problem.
- **Too Many Results** : A lot of mails can syntactically match the query. Hence the user has to spend a lot of time looking over many irrelevant mails before the desired one is found. There is no concept of sorting the results based on relevance here, as their is no measure of relevance. Either the words in query occur in the email or they do not.

A email dataset is a good example of a set of documents related to each other in a semantic way. Email threads, replies and forwarded mails share a common context and have a "latent" or hidden structure, which makes applying semantic search in this domain all the more apt.

2 Our Algorithm

Following the traditional literature on information retrieval, we will refer to the dictionary words as terms, and emails as documents (m in number). The algorithm has some steps which are performed offline and some which are performed online when the user gives a query. Each step is discussed in detail in the following sections. The outline of our algorithm is as follows:

- – When offline, run through all the documents in the mailbox to construct a list of terms occurring in documents. The terms are stemmed before being added to the list. Stop words are ignored. These are words like 'a', 'an', 'the', which have no classification value.
  – In addition, we maintain a list of unstemmed tokens. We do not remove those words which occur only once or twice (stop words are removed here as well).
- – Construct the term document matrix M. M_{ij} is the number of times term i occurs in document j. Adjust entries of M_{ij} based on the selected weighting scheme and get the a k-rank approximation: $M \approx M_k = U_k \Sigma_k V_k^T$. Store U_k, Σ_k and V_k in a file. This completes the offline steps.
  – Construct a binary term-document matrix, S, corresponding to the unstemmed token list.
- – When online take the input query and construct the query vector (q). Transform q to reduced space: $q_V = q^T U_k \Sigma_k^{-1}$. Dot-product q_V with the rows of V_k and output the results with the dot-products higher than a threshold.

– Construct q^{ns} such that $q_i^{ns} = 1$ if token i in the unstemmed list occurs in the query and 0 if not. Then, we return those documents whose dot-products with q^{ns} equal $\|q^{ns}\|^2$.

3 The Storyline: Implementation and Design Decisions

3.1 Does it Work?

We first implemented a naive version of LSI based on wordcount model and unprocessed emails (no stemming, no stop word removal, etc.) and tested whether dropping rank gets better results than the vector space model.

Consider the case of searching the mails using the query *reimbursement* in the `all_documents` folder of `scott-s` from the Enron dataset. There are several emails related to reimbursements which concern redressal of greviances and payment of compensation to the complainants for punitive damages. But some of these do not contain *reimbursement* syntactically. In place of reimbursement they use its synonyms: say compenstation. Therefore, our proposed method should find mails talking about reimbursements even though if they do not contain the word *reimbursement*.

We executed LSI for the search query *reimbursement*. To gauge the effect of the rank k on the results, we computed the dot-product of the search query q_V with V_k for various values of $k \in [1, m]$. We analyzed the top ranked results that the algorithm outputs.

- The most relevant email returned by the search result, i.e. the email with the highest dot-product value, contained the query term *reimbursement*. Its context was the same as the majority context of emails containing this term (i.e. greviance redressal).
- Emails containing *compensation* and based on the same context scored higher than those containing *reimbursement* but based on a different context. Emails containing *reimbursement* based on a non-majority context scored higher in relevance than other emails.
- As rank-approximation k was varied in $[1, c]$, the following trend was observed:
  - Dot-products of query q_V with the relevant emails increased upto a maximum value at some k_{max}, and decreased thereafter.
  - Dot-products of query q_V with the non-relevant mails followed the opposite trend: decreasing steeply till k_{max}, and either increasing or decreasing much slower thereafter.

  Overall, the dot-product of the search query q_V with V_k decreased substantially after k_{max}.

This trend was observed consistently across a wide range of queries. Figure 1 below depicts the trend of the dot-product of search query q_V with the most relevant mail (62), the most relevant mail containing synonym of the query (106), and an unrelated mail (1).

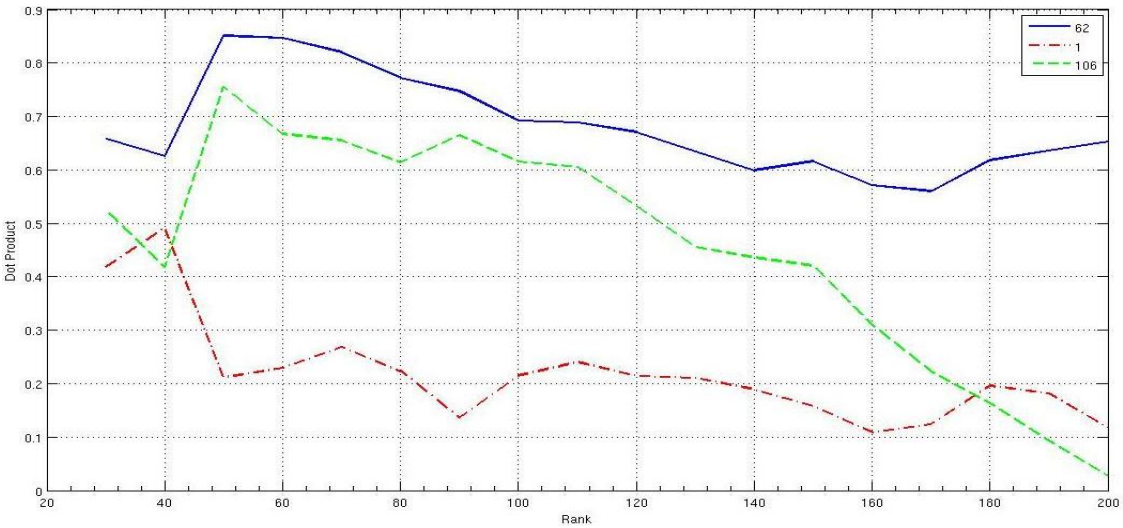


Figure 1: Dot Product as a function of the rank approximation k

As is evident from Figure 1, the proposed algorithm is successful and capturing the latent semantics of the email dataset and matching this against the query. For low values of k, we are underfitting. The $k < k_{max}$

dimension space is not sufficient to capture the structure of our data. When we increase k our results get better. But once we have enough features to capture the data if we add more features by increasing the rank then we are adding noise.

We got encouraging results from LSI: the documents had the structure for LSI to work.

3.2 Conditioning the Data

Since the dataset consisted of emails, there are bound to be abnormalities in the form of spelling mistakes or irrelevant terms occurring in spam. Hence we only consider the words occurring at least 3 times in the mailbox to be included as a term. We consider the stemmed form of the words as terms and we also stem the query. We used the Porter Stemmer described in [1]; it was more effective than the other stemmers we tried.

We remove the most frequently occurring words, called stop-words, like 'a', 'and', 'the', etc., which do not contribute to the semantics of the document. The stop word list consists of 119 most common stop words from literature. Then, we moved on to try different weight models.

3.3 Adding Weights

Proper term weighting can greatly improve the performance. We modified our initial wordcount model: M_{ij} was the number of times term i occurs in document j. The weighting scheme is composed of three different kinds of weights, the product of which gives us the relevant (Weighted) Term-Document matrix. Hence $M_{ij} = L_{ij}G_iN_j$.

3.3.1 Global Weights - G_i

G_i is the global weight for term i. [3] prescribes IGFL and IDFB weighting scheme for the Term-Document Matrix and IDFB for the query. It recommends IGFL over IDFB.

$$(G_i)_{IGFL} = \log(\frac{F_i}{n_i} + 1)$$

$$(G_i)_{IDFB} = \log(\frac{N}{n_i})$$

Here, F_i = total occurrences of term i in the email database n_i = number of documents in which term i occurs and N = total number of documents.

But in emails, some important terms occur in only a few mails. The user is frequently interested in such terms. But IGFL will give such important terms very low weights. From our experimentation, we found that IDFB outperforms IGFL by huge margins.

3.3.2 Local Weights - L_{ij}

L_{ij} is the local weight of term i in document j. [3] suggests that the SQRT and LOG (sublinear weights) schemes are good for this purpose. We tested both and found the behavior to be near identical for our purpose. We chose SQRT ($L_{ij} = \sqrt{f_{ij} - 0.5} + 1$, where f_{ij} is frequency of term i in document j) as our choice of local weight. The weight outperformed word count model for all queries. A comparison of Top-10 results with the term frequency case is given in Figure 2. The query is *court hearing*.

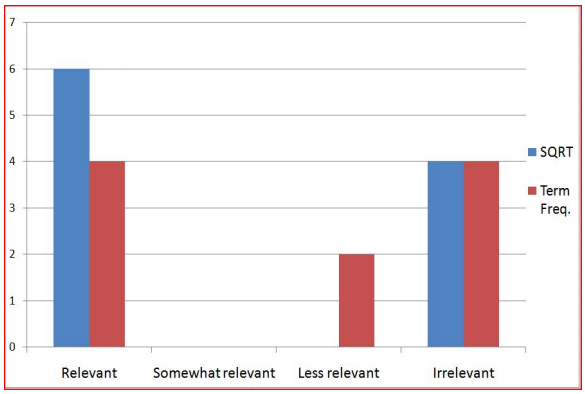


Figure 2: Comparison with local weights as SQRT and as wordcount. The global weight is IDFB

3.3.3 Normalization - N_j

N_j, the normalization factor for document j, attempts to compensate for the fact that documents are of different lengths. [3] suggested a COS scheme for normalization. This scheme works well for us.

$$N_j = \frac{1}{\sqrt{\sum_{i=0}^{m}(G_i L_{ij})^2}}$$

3.4 Heuristics for Further Improvement

3.4.1 Query Expansion With Synonyms

The user could type in a query which has a word that has not occurred in any email in the database. In such cases when a word cannot be found or when we want to implicitly increase the weight of a word in the query, we find a synonym for that word which exists in the present vocabulary and insert it into the query. We use `Nltk.word.corpus` for this. This helps improve the results marginally. Blue bars below represent the case when synonym is added.

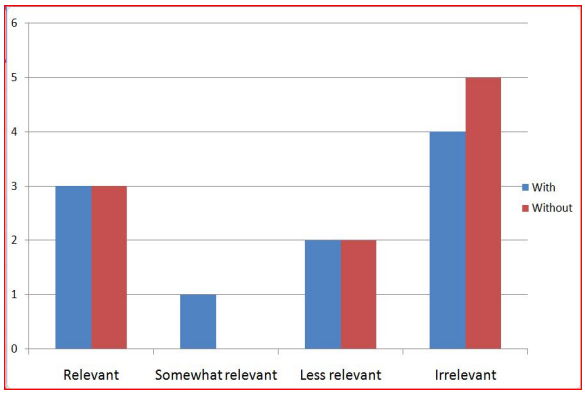


Figure 3: Query: "New Appointment" - Synonym Added : "assign"

3.4.2 Union with Syntactic Query

The results of the above algorithm may still not contain the desired mail. The reason lies in the conditioning steps mentioned in Section 3.2. Removal of words occurring less than 3 times in the database could cause the removal of many relevant terms. The user might remember that he had intentionally misspelled a word in a particular mail and he wants that particular mail.

This problem is overcome by including the mails which syntactically match the query exactly. The list of tokens considered for syntactic matching is not stemmed. We consider only those mails which have all the terms in a query. This "all-or-nothing" approach in conjunction with the semantic results covered almost all the relevant mails for any query.

3.4.3 Rocchio's Relevance Feedback Algorithm

For this algorithm, the user provides feedback about relevant and irrelevant documents. The approach proposes using the expanded query q'_V.

$$q'_V = \alpha q_V + \beta \frac{1}{\mid D_r \mid} \sum_{d_j \in D_r} d_j - \gamma \frac{1}{\mid D_{nr} \mid} \sum_{d_j \in D_{nr}} d_j$$

Here, D_{nr} is the list of irrelevant mails and D_r is the list of relevant mails. Setting the values of α, β and γ to be 1, 0.75 and 0.15 respectively and running the algorithm after giving it feedback improved results noticeably.

3.5 Helping LSI Learn

The user might opt to give feedback at times to improve the results but if user queries at a later time using the same query, it would be irritating to see the results that had been marked irrelevant previously still show up. So, it makes sense to record user feedback for future use. We do this by updating V_k. We tried out 2

methods of updating V_k. We used the following interpretation for the first one: there are latent variables called topics. Each term u_i is relevant to topic t by a quantity u_{it}. The relevance of each topic for the mailbox is s_{tt}. The relevance of document v_j to to topic t is given by v_{tj}. Hence for our weighted term document matrix M, $M_{ij} = \sum_t u_{it} s_{tt} v_{tj}$

The i^{th} coordinate of q_V indicates how important the topic i is to the user query. Hence we found the major topics the query q_V is related to, by taking the indices in q_V corresponding to the highest components (a cut-off was set here), and then reducing the components of the irrelevant documents in those topics.

The other method is to take the vector space approach and reduce the component of the irrelevant mail in the direction of q_V. So $d_j \in D_{nr}$ is updated as follows: $d'_j = d_j - \frac{1}{2}(d_j \cdot q_V)\frac{q_V}{\|q_V\|^2}$

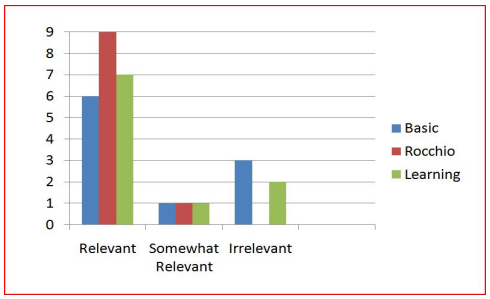
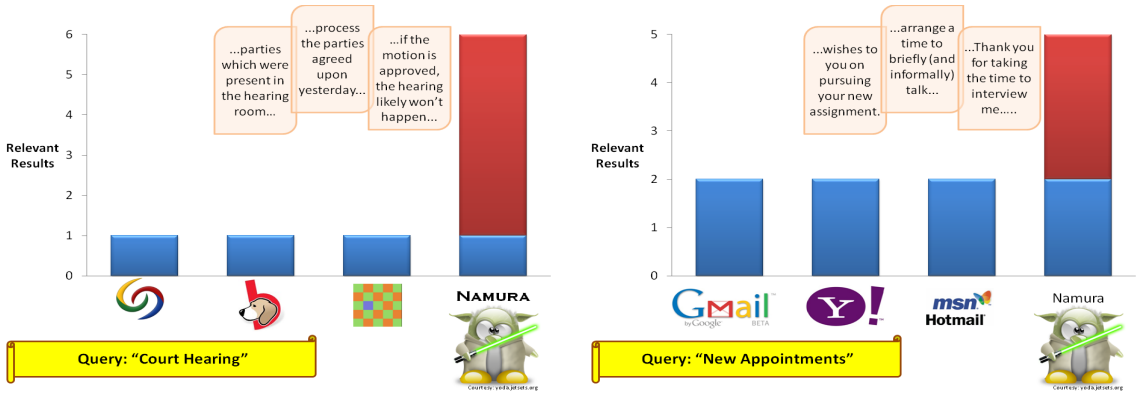


Figure 4: Top-10 Results for "Court Hearing" for the Basic, Rocchio-refined and Learning Schemes

4 Conclusion

Our search engine, **NAMURA**, when tested against existing desktop and email search engines returned improved results. The blue bar represents common results. The red bars show extra relevant results that we got.



(a) Against Desktop Search Engines

(b) Against Email Search Engines

Figure 5: Comparing Against Existing Search Engines

Our work clearly shows the importance of LSI-based techniques to email and desktop search.

5 References

1. Porter, M. "An algorithm for suffix stripping." Program 14.3 (1980): 130-137
2. Manning, C., Raghavan, P., Schutze, H. "Introduction to Information Retreival"
3. Erica Chisholm., Tamara G. Kolda, "New Term Weighting Formulas For the Vector Space Method in Information Retrieval"