

Generating Query Substitutions

Rosie Jones, Benjamin Rey and Omid Madani
Yahoo! Research
3333 Empire Avenue
Burbank, CA, 91504, USA
{jonesr,benjamir,madani}@yahoo-inc.com

Wiley Greiner*
Los Angeles Software Inc.
1329 Pine Street
Santa Monica, California 90405
w.greiner@lasoft.com

ABSTRACT

We introduce the notion of query substitution, that is, generating a new query to replace a user's original search query. Our technique uses modifications based on typical substitutions web searchers make to their queries. In this way the new query is strongly related to the original query, containing terms closely related to all of the original terms. This contrasts with query expansion through pseudo-relevance feedback, which is costly and can lead to query drift. This also contrasts with query relaxation through boolean or TFIDF retrieval, which reduces the specificity of the query. We define a scale for evaluating query substitution, and show that our method performs well at generating new queries related to the original queries. We build a model for selecting between candidates, by using a number of features relating the query-candidate pair, and by fitting the model to human judgments of relevance of query suggestions. This further improves the quality of the candidates generated. Experiments show that our techniques significantly increase coverage and effectiveness in the setting of sponsored search.

Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Query formulation, Search Process, Retrieval Models

General Terms

Algorithms, Experimentation

Keywords

Query rewriting, query substitution, paraphrasing, sponsored search

1. INTRODUCTION

Sometimes a user's search query may be an imperfect description of their information need. Even when the information need is well described, a search engine or information retrieval system may not be able to retrieve documents matching the query as stated. For example, a user issues the query "cat cancer", but all documents in the collection

*This work was carried out while this author was at Yahoo! Research

use the expression "feline cancer". Our setting is sponsored search, in which we attempt to match enormous numbers of queries to a much smaller corpus of advertiser listings. Here recall is crucial, as conjunctive search often leads to no matching results.

Existing solutions to this type of problem include relevance feedback and pseudo-relevance feedback, query term deletion [11], substituting query terms with related terms from retrieved documents [19], and Latent Semantic Indexing (LSI) [5]. Pseudo-relevance feedback involves submitting a query for an initial retrieval, processing the resulting documents, modifying the query by expanding it with additional terms from the documents retrieved and then performing a second retrieval with the modified query. Pseudo-relevance feedback has limitations in effectiveness [17]. It may lead to query drift, as unrelated terms are added to the query. It is also computationally expensive. Substituting query terms with related terms from retrieved documents also relies on an initial retrieval. Query relaxation or deleting query terms leads to a loss of specificity from the original query. LSI is also computationally expensive. In an alternative to automatic query modification or expansion, one could allow the user to select the appropriate related terms [1]. This allows for user-aided disambiguation, though also at the cost of an extra iteration and cognitive load.

We propose query modification based on pre-computed query and phrase similarity, combined with a ranking of the proposed queries. Our similar queries and phrases are derived from user query sessions, while our learned models used for reranking are based on the similarity of the new query to the original query as well as other indicators. We are able to generate suggestions for over half of all web search queries, even after eliminating sensitive queries such as those containing adult terms. A random sample of query pairs are labeled for training the models, and we find that the proportion of good suggestions is significant in the sample (even without any training): 55% of the suggestions in the sample are labeled as highly relevant. When we apply machine learning, we improve performance over the baseline significantly, improving the detection accuracy for highly relevant pairs to 79%. We describe the application of our methods to sponsored search, discuss coverage, and examine the types of useful suggestions and mistakes that our techniques make.

Others using user query sessions as a source of information to improve retrieval include Furnas [10], who explicitly asked users whether their reformulations should be used for indexing, and Radlinski and Joachims [15] who use query chains followed by clicks as a source of relevance ordering infor-

mation. Cucerzan and Brill [4] use query logs for spelling correction, but do not take advantage of user query reformulations.

Our contributions are: (1) identification of a new source of data for identifying similar queries and phrases, which is specific to user search data (2) the definition of a scheme for scoring query suggestions, which can be used for other evaluations (3) an algorithm for combining query and phrase suggestions which finds highly relevant phrases and queries 55% of the time, and broadly relevant phrases and queries 87.5% of the time and (4) identification of features which are predictive of highly relevant query suggestions, which allow us to trade-off between precision and recall.

In Section 2 we define the problem, describe our four types of suggestion quality, and discuss possible query rewriting tasks and the sponsored search application. In Section 3 we describe our technique for identifying related phrases (*substitutables*) based on user query-rewrite sessions, and the basic process for generation of candidate query substitutions. In Section 5 we describe a basic rewriting system, combining whole-query and phrase-substitutions based on user-session analysis. We then describe experiments using machine learning to improve the selection of query rewriting suggestions. In Section 6 we show that using machine-learned models improves the quality of the top-ranked suggestion, leading to high quality query substitutions, as measured through human evaluation. In Section 7 we show that we are able to assign a reliable confidence score to the query-suggestion pairs, which allows us to set thresholds and predict the performance of query suggestions when deployed.

2. PROBLEM STATEMENT

Given a query q_i , we wish to generate a modified query q_j which is related to the original query. We will write this: $q_i \mapsto q_j$ and refer to q_j as a *suggestion* for q_i .

There are many ways in which the new query q_j could be related to the original query q_i . Some may be improvements to the original query, others may be neutral, while others may result in a loss of the original meaning:

- same meaning, different way of expressing it
 - spelling change
 - synonym substitution (for example colloquial versus medical terminology)
- change in meaning
 - generalization (loss of specificity of original meaning)
 - specification (increase of specificity relative to original meaning)
 - related term

To accommodate and allow quantification of these and other changes, we define four types of query substitutions. We place these on a scale of 1 to 4, with the most related suggestions assigned a score of 1, and the least related assigned a score of 4. Examples of these are shown in Table 1, and we define these classes below.

2.1 Classes of Suggestion Relevance

1. **Precise rewriting:** the rewritten form of the query matches the user’s intent; it preserves the core meaning of the original query while allowing for extremely

minor variations in connotation or scope. Although the two forms may differ syntactically, in most cases, the rewriting process should be completely transparent to the user.

E.g.: automobile insurance \mapsto automotive insurance

2. **Approximate rewriting:** the rewritten form of the query has a direct close relationship to the topic described by the initial query, but the scope has narrowed or broadened or there has been a slight shift to a closely related topic.

E.g.: apple music player \mapsto ipod shuffle

3. **Possible rewriting:** the rewritten form either has some categorical relationship to the initial query (i.e. the two are in the same broad category of products or services, including close substitutes and alternative brands) or describes a complementary product, but is otherwise distinct from the original user intent.

*E.g.: eye-glasses \mapsto contact lenses,
orlando bloom \mapsto johnny depp*

4. **Clear Mismatch:** the rewritten form has no clear relationship to the intent of the original query, or is nonsensical. *E.g.: jaguar xj6 \mapsto os x jaguar*

2.2 Tasks

Depending on the task, different classes of rewriting may be acceptable.

- **Specific Rewriting (1+2)** If the goal is to have a closely related query, in order to retrieve new highly relevant results, only the 1s and 2s are acceptable. Here we will measure performance in terms of the proportion of {1,2}. We will refer to this as “specific rewriting” or “1+2”.
- **Broad Rewriting (1+2+3)** If the goal is to perform re-ranking of results retrieved with the initial query, (akin to relevance feedback), rewrites from classes 1, 2 and 3 may all be useful. Similarly all three classes may be useful if we wish to perform query expansion using the rewritten query. In addition, if the task is to find results relevant to the interests of the user, with the query as our indication of user interests, class 3 may be of interest too. Under all these conditions, we will measure our performance in terms of the proportion of query suggestions in classes {1,2,3}. We will refer to this as “broad rewriting” or “1+2+3”.

2.3 Query rewriting for sponsored search

In *sponsored search* [8], paid advertisements relevant to a user’s search query are shown above or along-side algorithmic search results. The placement of these advertisements is generally related to some function of the relevance to the query and the advertiser’s bid.

While in general web search there are often many documents containing all of the user’s search terms, this is not always true for sponsored search. The set of advertiser results is much smaller than the set of all possible web pages. For this reason, we can think of sponsored search as information retrieval over a small corpus, where in many cases a conjunctive match on all query terms will not give a result. In this setting, the ability to modify a user’s query (which

Class	Score	Examples	
Precise rewriting	1	automotive insurance	↔ automobile insurance
		corvette car	↔ chevrolet corvette
		apple music player	↔ apple ipod
		apple music player	↔ ipod
		cat cancer	↔ feline cancer
		help with math homework	↔ math homework help
Approximate rewriting	2	apple music player	↔ ipod shuffle
		personal computer	↔ compaq computer
		hybrid car	↔ toyota prius
		aeron chair	↔ office furniture
Possible rewriting	3	onkyo speaker system	↔ yamaha speaker system
		eye-glasses	↔ contact lenses
		orlando bloom	↔ johnny depp
		cow	↔ pig
		ibm thinkpad	↔ laptop bag
Clear mismatch	4	jaguar xj6	↔ os x jaguar
		time magazine	↔ time and date magazine

Table 1: Example queries and query-suggestions for each of the four classes.

may have no sponsored results) to a closely related query (which does have results) has extremely high utility.

Sponsored search has a few interesting constraints that we briefly touch on. The primary constraint is that we have a finite set of possible rewritings: the set of phrases for which we can return an advertisement. On one hand, this increases the difficulty of the task in terms of coverage, because even if we find a good rewriting for the initial query, it won’t necessarily be found in an advertiser’s listing. On the other hand, the constraint acts as a basic (language model) filter. If the rewritten form of the query is available for sponsored search, it is more likely to be meaningful. Thus this constraint can help filter out nonsensical rewritings (*E.g.*: *air jordan iv retro* ↔ *jordan intravenous 50s*). Another constraint relates to the classes of queries we can modify. For example, we avoid generating suggestions for sensitive queries such as those containing adult terms.

We expect that the work we present here is relevant to other applications of query rewriting, such as for general web retrieval.

3. SUBSTITUTABLES

Given an initial search query q_i , we wish to generate relevant queries q_j . We can do this either by replacing the query as a whole, or by substituting constituent phrases, as shown schematically in Figure 1.

Thus we would like to have a source of similar queries, and similar phrases. While we could use static sources such as WordNet [9] to identify similar phrases, in general this will not allow us to generate suggestions for new concepts such as products, movies and current affairs that arise in query streams. Another possibility is to use within-document co-occurrence statistics to find related phrases, as these have been found useful for finding query-relevant terms [19]. One could also use anchor text, which has been found useful for generating query refinements [12]. For using data as close as possible to our target task, however, we chose to work with user-sessions from search query logs. Previous work has shown these sessions to contain around 50% reformulations [11, 18]. In these query-session reformulations, a user modifies a query to another closely related query, through word

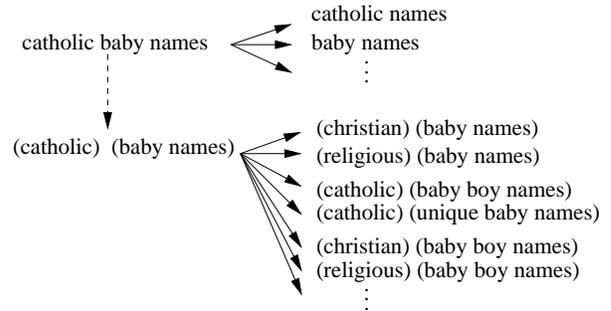


Figure 1: We generate multiple candidates Q' for query rewriting by both substituting for the whole query atomically, and by breaking it into phrases and generating substitutions for each phrase.

insertions, deletions and substitutions, as well as re-phrasing of the original query.

3.1 Definition of Query Pair

The data used comes from logs of user web accesses. This data contains web searches annotated with user ID and timestamp. A *candidate reformulation* is a pair of successive queries issued by a single user on a single day. Candidate reformulations will also be referred to as *query pairs*.

$$\text{candidateQueryPairs}(user_i, day_j) = \{ \langle q_1, q_2 \rangle : (q_1 \neq q_2) \wedge \exists t : \text{query}_t(user_i, q_1) \wedge \text{query}_{t+1}(user_i, q_2) \}$$

We collapse repeated searches for the same terms, as well as query pair sequences repeated by the same user on the same day. We then aggregate over users, so the data for a single day consists of all candidate reformulations for all users for that day.

3.2 Phrase Substitutions

Whole queries tend to consist of several concepts together, for example “(new york) (maps)” or “(britney spears) (mp3s)”. Using phrases identified by high point-wise mutual information, we segment queries into phrases (where a single word

can be a phrase), and find query pairs in which only one segment has changed. For example, the query pair : (britney spears) (mp3s) \rightarrow (britney spears) (lyrics) gives us both an instance of a whole query pair, as well as the pair: **mp3s** \rightarrow **lyrics**. We then identify this pair of phrases as a candidate phrase pair.

Note that the point-wise mutual information measure we use to identify phrases looks for adjacent terms whose mutual information is above a threshold:

$$\frac{P(\alpha, \beta)}{P(\alpha)P(\beta)} > \kappa$$

where we set the threshold κ to be 8. We also experimented with the *connectivity* approach to identifying phrases [16], and found that our methods are not sensitive to the approach used to identify phrases.

3.3 Identifying Significant Query Pairs and Phrase Pairs

In order to distinguish related query and phrase pairs from candidate pairs that are unrelated, we use the pair independence hypothesis likelihood ratio. This metric tests the hypothesis that the probability of term q_2 is the same whether term q_1 has been seen or not, by calculating the likelihood of the observed data under a binomial distribution using probabilities derived using each hypothesis [7, 13].

$$H_1 : P(q_2|q_1) = p = P(q_2|\neg q_1)$$

$$H_2 : P(q_2|q_1) = p_1 \neq p_2 = P(q_2|\neg q_1),$$

The likelihood score is

$$\lambda = \frac{L(H_1)}{L(H_2)}$$

The test statistic $-2 \log \lambda$ is asymptotically χ^2 distributed. Therefore we work with the log likelihood ratio score:

$$\text{LLR} = -2 \log \lambda = -2 \log \frac{L(H_1)}{L(H_2)}$$

A high value for the likelihood ratio suggests that there is a strong dependence between term q_1 and term q_2 . In Table 2 we show examples of phrases with high likelihood ratios for the phrase or query “dog”. We observe that many semantic relationships are captured in these highly correlated queries and terms. We refer to query pairs and phrase pairs above a threshold for the LLR score as *substitutables*. In related work we are evaluating these related terms for performance on standard semantic similarity tests. We will see in Section 6 that they perform well for generating similar queries.

Because of the χ^2 distribution of λ , a score of 3.84 for LLR gives us a 95% confidence that we can reject the null hypothesis, and two phrases are statistically significantly related. However, this will give us 1 in 20 spurious relationships. As we are dealing with millions of phrases, we set the threshold on LLR much higher, generally to above 100, based on observation of the substitutable pairs.

3.4 Generating Candidates

We seek to generate statistically significant related queries for arbitrary input queries. For frequent queries, we have dozens of such related queries. But for less frequent queries, we may not have seen sufficiently many instances of them to

dog \rightarrow dogs	9185	(pluralization)
dog \rightarrow cat	5942	(both instances of ‘pet’)
dog \rightarrow dog breeds	5567	(generalization)
dog \rightarrow dog pictures	5292	(more specific)
dog \rightarrow 80	2420	(random junk or noise)
dog \rightarrow pets	1719	(generalization – hypernym)
dog \rightarrow puppy	1553	(specification – hyponym)
dog \rightarrow dog picture	1416	(more specific)
dog \rightarrow animals	1363	(generalization – hypernym)
dog \rightarrow pet	920	(generalization – hypernym)

Table 2: Terms and queries which can be substituted for the term or query “dog”, along with likelihood ratios, based on user query rewriting sessions. The semantic relationship is shown for explanatory purposes only.

have any statistically significant related queries. We can’t ignore infrequent queries: infrequent: the power-law (Zipf) distribution of query terms leads to a large proportion of rare queries.

In the same way as we generated the phrase-substitutables, we can break up the input query into segments, and replace one or several segments by statistically significant related segments. This will help cover the infrequent queries. Figure 1 shows a schematic of the approach we take:

- generate m candidate whole-query substitutions $q_i \mapsto q_{i_1}, q_{i_2} \dots q_{i_m}$
- segment query into phrases $p_1 \dots p_n$
- for each phrase p_i
 - generate k phrase-substitutions $p_i \mapsto p_{i_1}, p_{i_2} \dots p_{i_k}$
 - generate new query from a combination of original phrases and new phrases: $q_i \mapsto p_1 \dots p'_j \dots p_n$

This gives us a set of query-substitution candidates which we will denote Q' .

As an example, consider the query “catholic baby names”. We can find whole-query substitutions from our query substitutables, such as “catholic names”. We can also segment the query into the two phrases “catholic” and “baby names”, and find substitutions for these phrases (see Figure 1).

4. EXPERIMENTAL METHOD

We will compare a variety of methods for generating query substitutions. Our evaluation will consider only the query suggestion, *i.e.*, given a randomly sampled query q_i , we will assess the quality of the query substitution q_j we suggest. We do not assess end-to-end retrieval performance.

For each evaluation, we use a random sample of 1000 initial queries q_i sampled from query logs from a disjoint time period to our substitutables query data, and generate a single suggestion q_j for each. We will evaluate the accuracy of approaches to choosing the suggestion to generate, by reporting the proportion of suggestions falling into the classes **precise (class 1+2)** and **broad (class 3+4)**.

To assess whether there are properties of the suggestions which are predictive of good quality, we will train a machine-learned classifier on the labeled $\langle \text{query}, \text{query-suggestions} \rangle$ pairs. We will then evaluate our ability to produce suggestions of higher quality for fewer queries, by plotting precision-recall curves.

5. AUTOMATIC ASSESSMENT OF SUBSTITUTION QUALITY

Whole query substitutions and many of the possible combinations of substituted phrase segments yield hundreds of candidates for common queries. Only a few are good, and we need to assess and predict quality as well as rank.

We first describe two simple methods for ranking candidates. In order to develop a more sophisticated ranking scheme, we take the top suggestion from one of these ranking schemes and use machine learning and human judgements to learn a model of high quality suggestions. We describe how we use the learned classifier to learn a more accurate re-ranking of candidate suggestions in Section 6.2.

5.1 Basic Query Substitution Ranking Algorithms

5.1.1 Random Ranking Algorithm

Our baseline candidate selection method is *randomRank*:

1. Set the maximum number of whole query alternatives m to 10
2. Segment the query, and assign number of alternatives per phrase, k , as a function of the number of constituent phrases n :
 - if $(n > 5)$ $k = 5$
 - else $k = 10$

For **randomRank** we require all whole query and phrase suggestions to have an LLR score of at least 50. We set the threshold lower than the value of 100 we have empirically observed to be a useful setting, in order to assess the performance of suggestions over a range of LLR scores.

We sample one query uniformly at random from this sample. Note that we are much more likely to select a phrase-substitution than a whole query substitution: for a query with two phrases, there are up to $11 \times 11 - 1 = 120$ new queries generated by phrase substitution, and only 10 generated by whole-query substitution.

5.1.2 Substitution Type and Log-likelihood Ratio Score

We designed our next method, *LLRNumSubst*, using the following intuitions, based on our experience with samples of the generated candidates:

1. First try whole-query suggestions ranked by LLR score
2. then try suggestions which change a single phrase, ranked by the phrase substitution LLR score
3. then try suggestions which change two phrases, ordered by the LLR scores of the two phrases
4. ...

For more efficient processing of candidates, we constrain the number of suggestions by limiting suggestions per phrase as shown in Table 3. Very long queries seldom have a sponsored search result and we did not attempt to generate candidates for them. We require all query and phrase suggestions to have an LLR score of at least 100.

Num. phrases	Max. suggestions / phrase
1	99
2	9
3	2
4,5	1
6+	0

Table 3: For the *LLRNumSubst* ranking approach to generating suggestions, limits on suggestions per phrase are applied, depending on the number of phrases in the query. The goal is to keep the total number of suggested queries around 100.

5.2 Query Substitution with Machine Learning

Rather than making assumptions about what makes a good substitution, we can treat our problem as a machine learning problem. The target to learn is the quality of the substitution, and we provide features that we expect to be reasonably efficiently computable. We tried both linear regression and binary classification approaches. For classification, we learn a binary classifier over query pairs:

$$g(q_i, q_j) \mapsto \{+1, -1\} \quad (1)$$

We evaluate on two binary classification tasks, as specified in Section 2.2:

- **broad (classes 1+2+3)** for which the negative class will be rewritings labeled 4
- **specific (classes 1+2)** for which the negative class will be rewritings labeled 3+4.

5.2.1 Labeled Data

Our training data was a sample comprised of the top-ranked suggestion for each of 1000 queries, where the top-ranked suggestion was generated using the **LLRNumSubst** ranking scheme described Section 5.1.2. While using random suggestions from the **randomRank** scheme as training data would lead to a more general model, using **LLRNumSubst** suggestions allows us to pre-filter for high-likelihood suggestions, and then learn a ranking among them. We had the pairs $\langle q_1, q_2 \rangle$ of query and suggestion manually labeled on the scale of 1 to 4 as described in Section 2.

5.2.2 Features

We generated 37 features for each initial and rewritten query pair (q_1, q_2) . These features were of 3 types:

- Characteristics of original and substituted query in isolation: length, number of segments, proportion of alphabetic characters.
- Syntactic substitution characteristics: Levenshtein edit distance, number of segments substituted, number of tokens in common, number of tokens specific to q_1 , to q_2 , size of prefix overlapping, stemming relationship.
- Substitution statistics: LLR, frequency, $p(q_2|q_1)$, mutual information. Where multiple phrases were substituted in a single query, these features were computed for the both the minimum and the maximum.

A complete list of these features is shown in Table 4.

Feature	Description
length	number of characters (q_1)
nLetters	number of letters (a-zA-z) in q_1
nTokens	number of words in q_1
tok1	number of words only in q_1
tok2	number of words only in q_2
nPhrases1	number of phrases in q_1
nPhrases2	number of phrases in q_2
sponsoredResults	q_1 has a sponsored result
numSubst***	number of phrases substituted
tok12	number of words in common
Word edit-dist***	proportion of words changed
nPhrasesDiff	$nPhrases1 - nPhrases2$
distr***	normalized edit distance
dist	edit distance
possibleStem	is q_2 a stem of q_1
prefixOverlap	num. prefix characters overlapping
LLR1min	$\min(\text{LLR}(p_i \mapsto p_j))$
LLR2min	$\min(\text{LLR}(p_j \mapsto p_i))$
freq1min	$\min(\text{freq}(p_i \mapsto p_j))$
freq2min	$\min(\text{freq}(p_j \mapsto p_i))$
f1divf2min	$(\text{freq1min} + 0.5) / (\text{freq2min} + 0.5)$
dp1min	smallest deletion probability in q_1
dp2min	smallest deletion probability in q_2
prP2givenP1min	$\min(p(p_i \mapsto p_j))$
prP1givenP2min***	$\min(p(p_j \mapsto p_i))$
mutInfmin	$\min(\text{MI}(p_i, p_j))$
LLR1max	$\max(\text{LLR}(p_i, p_j))$
LLR2max	$\max(\text{LLR}(p_j, p_i))$
freq1max	$\max(\text{freq}(p_i \mapsto p_j))$
freq2max	$\max(\text{freq}(p_j \mapsto p_i))$
f1divf2max	$(\text{freq1max} + 0.5) / (\text{freq2max} + 0.5)$
dp1max	max deletion probability in q_1
dp2max	max deletion probability in q_2
pu2givenu1max	$\max(p(p_i \mapsto p_j))$
pu1givenu2max***	$\max(p(p_j \mapsto p_i))$
mutInfmax	$\max(\text{MI}(p_i, p_j))$
$\prod_i p(u_i \mapsto u'_i)$	

Table 4: Features considered for query rewriting. Those marked with ‘*’ were significant at the 0.001 level.**

5.2.3 Linear Regression

We used the original labels {1, 2, 3, 4} and performed standard linear regression. The forward-backward stepwise process reduced the number of features from 37 to 20. We kept only the features with p-value smaller than 5×10^{-4} plus *number of substitutions* and experimented with different combinations. The simplest best fit was obtained with the following features:

- Word distance: prefer suggestions with more words in common with the initial query
- Normalized edit distance: prefer suggestions with more letters in common with the initial query
- Number of substitutions: prefer whole query suggestions over phrase suggestions, prefer fewer phrases changed.

It is interesting to note that none of the features relating to the substitution statistics appeared in the best models. This may be due to the fact that the training query suggestions were selected using the LLRNumSubst method which takes LLR score into account. The ranking function we learn is shown in Equation 2.

$$\begin{aligned}
 f(q_1, q_2) = & 0.74 + 1.88 \textit{editDist}(q_1, q_2) \\
 & + 0.71 \textit{wordDist}(q_1, q_2) \\
 & + 0.36 \textit{numSubst}(q_1, q_2) \quad (2)
 \end{aligned}$$

We can interpret this model as saying that, given that a suggestion is among the top ranked suggestions according to the LLRNumSubst ranking, we should prefer query substitutions with small edit distance (perhaps spelling and morphological changes) and with small word edit distances (perhaps word insertions or deletions). We should prefer whole-query suggestions, and if we substitute at the phrase segment level, the fewer the substitutions the better.

5.2.4 Classification Algorithms

We experimented with a variety of machine learning algorithms to assess performance achievable and the utility of the overall feature set. We experimented with linear support vector machines (SVMs) [3] allowing the classifier to choose its own regularization parameter from a fixed set (using portion of training data only), and decision trees (DTs). For the SVM, we normalized features by dividing by the maximum magnitude of the feature, so all features were in the range [-1,1]. For tree induction, no pruning was performed, and each feature was treated as numeric. Bags of 100s of decision trees performed best, under a number of performance measures. On the task of distinguishing {1+2} from {3+4} (baseline of 66% positive using data generated with LLRNumSubst described in Section 5.1.2), the zero-one error using such committees of trees was just above 0.21 (40 10-fold cross-validation trials). Note that the error rate of baseline (majority classifier) is 0.34. For other more restricted classifiers such as linear SVMs and single trees we obtained error rates of around 0.24. The following single tree of depth only two, induced by treating features as boolean (zero or nonzero), was sufficient to obtain this level of zero-one performance.

```

If (number of tokens in common is nonZero )
  then {1+2}
else if (prefix overlap is nonZero)
  then {1+2}
  else
    {3+4}
  end
end

```

We can interpret this model as saying that a suggestion should ideally contain some words from the initial query, otherwise, the modifications should not be made at the beginning of the query.

6. RESULTS

For all learning algorithms we report on, we performed 100 random 90-10 train-test splits on the labeled data set, and accumulated the scores and true labels on the hold-out instances. We plot the precision-recall curve to examine the trade-off between precision and recall. The F_1 score is the harmonic mean of precision and recall: $F_1 = \frac{2PR}{P+R}$, and Max-F refers to the maximum F_1 score along the precision-recall curve. The breakeven point (BP) is the precision at the point where the precision and recall are equal (or closest). Finally, average precision is the average of precision scores at every recall point. It serves as a useful summary statistic capturing performance across the entire precision-recall curve.

6.1 Results of Regression and Classification

Figure 2 shows that the precision recall curves for four learning algorithms: SVM, linear (regression) model, 2 level decision tree, and bag of decision trees. Interestingly, they all converge around precision of 76% and recall of 92%, which appears also to be their max-F point. They have roughly the same max-F measure (83-84% - see Table 5). The shape of the curve is different for each model. The 2 level decision tree does not perform well overall. As the decision tree splits the possible scores into only 3 regions,

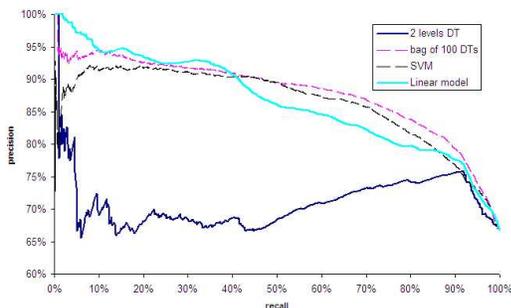


Figure 2: Precision - recall for linear model, 2 levels DT, bag of 100 DTs, SVM

	BP	max-F	average precision
2 level DT	0.71	0.83	0.71
bag of 100 DTs	0.83	0.84	0.88
SVM	0.81	0.83	0.86
linear model	0.80	0.84	0.87
baseline	0.66	0.66	0.66

Table 5: break even point - average precision - max-F for 2 levels DT, bag of 100 DTs, SVM, linear model

the precision recall curve is not convex. The linear regression model, using only 3 features, has the best precision for low levels of recall, but the bag of decision tree is better for recall between 40 and 90%. Overall, the bag of decision trees performs the best under a number of performance measures (zero-one accuracy and curve related measures such as precision-recall breakeven-point, average precision, and area under ROC curve, as shown in Table 5.)

We would like to apply one of these models for selecting the suggestion for an initial query from the pool of possible candidates, Q' . For our subsequent experiments we experimented with the linear regression model for its execution time efficiency, as it only requires three easy to compute features.

6.2 Precision of Top-ranked Suggestion

We will now turn our attention from the binary classification of pre-selected query pairs, to the ranking of multiple candidate suggestions Q' for a single input query. We have the two ranking schemes (1) **randomRank** and (2) **LLR-NumSubst** described in Sections 5.1.1 and 5.1.2. We also have the linear model given in Equation 2, which was trained on queries generated with **LLR-NumSubst**. We use this linear model to score and re-rank suggestions produced by **LLR-NumSubst** on a new set of 1,000 randomly selected queries, and take the top-ranked suggestion as the candidate. We refer to this algorithm as the linear model, or (3) **NumSubstEdit**.

We can now compare the precision of the top-ranked suggestion of these three ranking schemes. Note that each was used to generate suggestions for a distinct sample of 1,000 queries. In Table 6 we see that the substitutables are an excellent source of related terms: the baseline random ranking achieved precision of 55% on the **specific rewriting** task of identifying highly relevant suggestions for queries. The **LLR-NumSubst** algorithm improved the performance to 66%,

Ranking Scheme	{1+2}	{1 + 2 + 3}
random	55%	-
LLR	6%	87.5%
numSubstEdit	74%	87.5%

Table 6: Precision of top-ranked suggestion for each ranking scheme for precise (1+2) and broad (1+2+3) rewriting.

while **NumSubstEdit**'s re-ranking of a separate sample of **LLR-NumSubst** candidates increased precision of the top suggestion to 74%.

For generating a suggestion for **broad rewriting**, which allows suggestions from each of classes $\{1,2,3\}$ **LLR-NumSubst** generates an appropriate top-ranked suggestion for 87.5% of a sample of 1000 queries. Our source of data, the substitutables, is by construction a source of terms that web searchers substitute for one another, leading to a large proportion of broadly related suggestions. Re-ranking **LLR-NumSubst**'s candidate suggestions with **NumSubstEdit** does not result in increased precision of the top-ranked candidate for **broad rewriting**.

6.3 Coverage

We are not able to generate suggestions for all queries for several reasons:

- Data sparseness: for many queries, we have seen too few instances to have statistically significant whole query or phrase substitution instances.
- Sponsored search setting: we are constrained to generating suggestions for which we have an advertiser.
- Risk averse algorithm design: we eschew generating suggestions for adult queries (around 5% of the input query sample), and for other sensitive terms.
- Novelty constraint: we reject suggestions which are deemed to be identical to the original query by a baseline normalization procedure.

Under these constraints, our methods (e.g., Random Selection of Section 5.1.1) yield at least one suggestion for about 50% of queries.

When we break down the coverage by query decile we see that the coverage varies by decile. Decile 1 contains the queries constituting the top 10% of query volume, while decile 10 contains the most infrequent queries, most of which have a search frequency of only 1 in the sampled time-period. The histogram shown in Figure 3 was generated by using the **NumSubstEdit** method to select a candidate. We see that our method allows generation of suggestions for both frequent and infrequent queries.

We generate a suggestion for more than 10% of queries from decile 10, many of which would typically have no sponsored search result. This incremental coverage is primarily due to phrase substitution: for the frequent queries, we were able to find whole query substitutions, but for the less frequent ones, only breaking the query into phrase segments allowed us to generate a result.

Examples of queries with a low monthly query frequency for which we changed 1 or 2 units:

new testament passages \mapsto *bible quotes*
cheap motels manhattan, ny \mapsto *cheap hotels manhattan, ny*

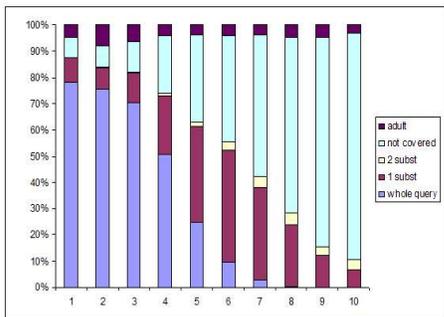


Figure 3: Coverage per decile, per type of substitution

7. CONFIDENCE MODEL

For each query suggestion generated by our algorithm, we would like to associate a probability of correctness (confidence score). For example, for the query pair 1943 nickel \mapsto 1943 dime, our `numSubstEdit` linear model assigned the score 2.27, which we can interpret as close to a precise match (1+2). To convert this to a probability we could normalize by shifting and scaling, setting the linear model score 1 to a probability of correctness of 1, and the linear model score 4 to a probability of correctness of 0. This would give a probability of correctness to this example of 58%. Depending on the distribution of scores, we may be able to give more reliable confidence estimates than this. In this section we describe our experiments on turning the output of the linear model into confidence scores.

7.1 Evaluating Confidence Models

We evaluate our confidence models by comparing them to the true binary (0,1) label. If a query pair was labeled 1 or 2 by our human labelers, the ideal confidence model would say the pair is correct with probability 1. In this case we will say the true probability of correctness p_{true} is 1. If the query pair was labeled 3 or 4 by our human labelers, the ideal confidence model would say the pair is correct with probability 0. In this case we will say the true probability of correctness p_{true} is 0.

We evaluate our confidence models using root-mean squared error (RMSE):

$$RMSE = \sqrt{\frac{\sum (p_{model} - p_{true})^2}{n}}$$

where n is the number of query pairs we test the model on. We also evaluate log-loss, which uses the log of a linear loss function.

7.2 Baseline Confidence Models

Our **uniform** model assumes the probability is the same for all suggestions: the overall precision of the model (0.73). **Shift and scale** (SS) is the simple monotonic transformation which maps the score onto a [0,1] scale by subtracting the minimum score and dividing by [max - min].

7.3 Comparison of Confidence Estimation Techniques

We experimented with several well known techniques for transforming a score into a probability, and evaluated their

	Uniform	SS	Sigm	Iso	ADist
RMSE	44.4%	41.7%	40.7%	40.8%	41.2%
LogLoss	84.1%	76.3%	73.3%	74.6%	74.9%

Table 7: Error at estimating the probability of correctness for query suggestions, by fitting a function from the linear model score to an estimate of the probability of correctness.

RMSE and Log-Loss on 100 80-20 train-test splits. Results are shown in Table 7.

Isotonic regression [6] (Iso) performs well, but tends to overfit¹. Fitting an asymmetric distribution [2] (ADist) is a promising technique in general, but for our task no true distribution could model well the {3 + 4} class and thus the performance was not as good as with the other techniques. The best results were obtained with a sigmoid [14] (SS). The sigmoid also has the advantage of being a simple model to implement.

Consequently, we used the sigmoid to model the probability function:

$$P(\text{label}(q_i, q_j) = \{1, 2\} | f) = \frac{1}{1 + \exp(1.85 f(q_i, q_j) - 4.9)}$$

7.4 Threshold Model

For different applications of our query-suggestion algorithm, we may want to ensure that the average precision will be higher than a given threshold. This differs from the confidence model, in that we consider a set of <query,suggestion> pairs, and not just one <query,suggestion> pair. For a set of pairs, if we accept only the suggestions with a confidence higher than a given threshold, then the average precision of the set is at least as high as the precision at the threshold itself, and will probably be significantly higher.

We could directly rely on the observed data, and for each value of the threshold, count how many times suggestions with a score higher than this threshold were correct. A drawback of this approach is that for high thresholds, we will have very little data with which to estimate the precision. This can be observed on the graph of the estimated precision for given thresholds in Figure 4. The confidence interval increases with the threshold, and with a high threshold, we are not able to give a reliable estimate.

We have a probability model, and we can take advantage of it using the following integration. For a set of query transformations Qt , if we accept only those with a probability of correctness Pc greater than T , the average precision will be:

$$Precision(Qt | Pc(\text{class}(qt_i) = \{1 + 2\}) > T) = \frac{\int_{x=\text{sigm}^{-1}(T)}^1 \text{sigm}(x) dp(x)}{\int_{x=\text{sigm}^{-1}(T)}^1 dp(x)}$$

where sigm is the transformation of the sigmoid of the confidence model and $p(x)$ is given by the histogram of the score. The model fits the observed data quite well, up to a threshold of 80%. After that, it seems conservative. This may be useful as it prevents us from creating an algorithm which promises 100% precision and delivers only 90% precision.

¹The variance of the RMSE across different cross validation runs was much higher than for other techniques.

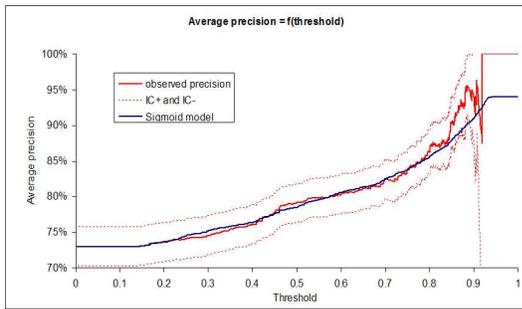


Figure 4: Average precision as a function of a threshold on the confidence of individual query suggestions.

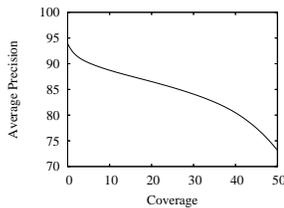


Figure 5: Precision Coverage trade-off

Still it is odd that we have an under-confident estimate. This is due to the fact that the true model of the probability is not sigmoid-shaped for high scores. When we replace the flattening part of the sigmoid with a line ramping up faster than the sigmoid, we have better results as measured with RMSE. Nevertheless, since we have few datapoints with high scores, it is safer to assume that the probability for high scores is flattening. We can integrate to get the associated confidence threshold. For instance, if we want to ensure an average precision of 75%, we need to reject any rewriting with a confidence lower than 17%. If we want to ensure an average precision of 85%, we need to refuse any rewriting with a confidence lower than 76%.

In Section 6 we looked at precision-recall curves. We now take into account both the incomplete coverage, the continuous score of the linear model and the threshold model transformation. This allows us to tune for more or less coverage and predict the expected average precision. As we can see in Figure 5, we can have, for example:

- precision: 90% coverage 7%
- precision: 80% coverage 42%
- precision: 75% coverage 49%

7.5 Examples of Estimated Relevance Scores

Using the NumSubstEdit linear model with sigmoid normalization, we have observed that substitution types in order of decreasing confidence tend to be spelling variants, then query expansion and relaxation, then synonyms and related terms, and finally approximate matching with every word changed. In Table 8 we see example queries along with their suggestions, and the relevance score assigned by our model.

8. DISCUSSION

Initial query	Rewritten query	label	\hat{P}
anne klien watches	anne klein watches	1	92%
sea world san diego	sea world san diego tickets	2	90%
restaurants in washington dc	restaurants in washington	2	89%
boat insurance cost?	boat insurance price	1	79%
nash county	wilson county	3	66%
frank sinatra	elvis presley	4	17%
birth certificate	birth	4	22%
ampland	google	4	22%

Table 8: Example queries with the suggestions generated by our algorithm, along with manually assigned labels (1=close rewriting, 4=bad rewriting) and the estimated probability of relevance assigned by our confidence model.

meaning of dreams	↦	interpretation of dreams (synonym)
furniture etegere	↦	furniture etagere (spelling correction)
venetian hotel	↦	venetian hotel las vegas (expansion)
delta employment credit union	↦	decu (acronym)
lyrics finder	↦	mp3 finder (related term)
national car rental	↦	alamo car rental (related brand)
amanda peet	↦	saving silverman (actress in)

Table 9: Example specific and broad suggestions generated by NumSubstEdit. Many semantic relationships are found in the rewrites (shown here for illustrative purposes only).

The NumSubstEdit algorithm prefers suggestions with small edit distance and few words changed. This is often effective for identifying **specific (1+2)** suggestions, as we showed in Section 6.2, but also generates **broad rewrites**. Examples of both are shown in Table 9.

One of the causes of the broader suggestions is our approach to identifying possible phrase substitutions. Very general contexts can lead to broad types of substitutions: (britney spears) (mp3s) \rightarrow (christina aguilera) (lyrics) gives us the phrase pair: **britney spears** \rightarrow **christina aguilera**. We are investigating methods for detecting very broad changes, such as considering the entropy of the rewrites found in a given context.

While we showed in Section 6.3 that coverage varies across deciles, the precision of suggestions is generally constant across deciles. However, there are still differences between the suggestions we generate for frequent and infrequent queries. We tend to generate more spelling variants for the infrequent queries: we see 0% spelling change for initial queries in decile 1, and 14% for decile 10. The reason may simply be that infrequent queries are much more likely to contain misspellings, and spelling corrections are the easiest suggestions to generate.

Even with an excellent source of related terms such as the substitutables, we occasionally generate poor suggestions. We have observed that these errors are frequently produced because of polysemy: some terms are good substitutes for each other in one context, but not in another. For instance, while *software* is related to *system* in the context of computer science, it is not in the context of the term “aerobic”. We generated the poor suggestion:

aerobic system \mapsto *aerobic software*

Another problem which occurs frequently is in the rewriting of people’s names. Two similar first names could possibly be used to refer to the same person, but if given with a last name, they are not equivalent anymore. For instance:

andre wilson \mapsto *andrew wilson* (different person)

Another source of mistakes is when the relationship between the initial term and its substitute exists, but is too weak. Then the substitution provokes a shift that is too distant in the meaning from the initial query. For instance:

craig's list \mapsto *monster*

Both are popular queries, certainly likely issued often in the same sessions, but it is unlikely that a user's specific search need when searching for one is satisfied by serving results for the other.

Overall our query substitution technique is extremely efficient, especially in comparison to retrieval-based techniques such as pseudo-relevance feedback, and matrix multiplication methods such as LSI. For whole-query suggestions, we are able to precompute the query substitutions and their scores offline, and so at run-time we require just a look-up. For phrase substitutions, we precompute edit distance between phrases offline, so when we look-up substitutions for each phrase at run-time, we require linear normalization of the edit-distance score, as well as computing the linear score with multiplications and additions.

9. CONCLUSIONS AND FUTURE WORK

We have shown that we are able to generate highly relevant query substitutions. Further work includes building a semantic classifier, to predict the semantic class of the rewriting. With such a classifier we would be able to focus on the targeted subtypes of rewriting, such as spelling variants, synonyms, or topically related terms.

To improve our algorithm, we can also take inspiration from machine translation techniques. Query rewriting can be viewed as a machine translation problem, where the source language is the language of user search queries, and the target language is the language of the application (for instance advertiser language in the case of sponsored search).

In order to generalize our work to any application, we also need to work on introducing a language model, so that in the absence of filtering with the list of sponsored queries, we avoid producing nonsensical queries. In addition, with the algorithm in operation we could learn a new ranking function using click information for labels.

10. ACKNOWLEDGEMENTS

Thanks to Tina Krueger and Charity Rieck for work on the evaluation, to Paul Bennett for helpful discussions, and to the anonymous reviewers for helpful comments.

11. REFERENCES

- [1] P. Anick. Using terminological feedback for web search refinement - a log-based study. In *Proceedings of the Twenty-Sixth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-2003)*, 2003.
- [2] P. N. Bennett. Using asymmetric distributions to improve text classifier probability estimates. In *SIGIR '03: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 111–118, New York, NY, USA, 2003. ACM Press.
- [3] C.-C. Chang and C.-J. Lin. *LIBSVM: A Library for Support Vector Machines*, 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [4] S. Cucerzan and E. Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of EMNLP 2004*, pages 293–300, 2004.
- [5] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
- [6] L. Dumbgen. Pair-adjacent violators (PAV), available at <http://www.math.mu-luebeck.de/workers/duembgen/software/software.html>. In *Statistical Software (MATLAB)*, 2000.
- [7] T. E. Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993.
- [8] D. C. Fain and J. O. Pedersen. Sponsored search. In *Bulletin of the American Society for Information Science and Technology*, 2005.
- [9] C. Fellbaum. *WordNet: An Electronic Lexical Database*. The MIT Press, 1998.
- [10] G. W. Furnas. Experience with an adaptive indexing scheme. In *CHI '85: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 131–135, New York, NY, USA, 1985. ACM Press.
- [11] R. Jones and D. C. Fain. Query word deletion prediction. In *Proceedings of the Twenty-Sixth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-2003)*, pages 435–436, 2003.
- [12] R. Kraft and J. Zien. Mining anchor text for query refinement. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW-2004)*, pages 666–674, 2004.
- [13] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
- [14] J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.
- [15] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *KDD '05: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 239–248, New York, NY, USA, 2005. ACM Press.
- [16] K. M. Risvik, T. Mikolajewski, and P. Boros. Query segmentation for web search. In *Poster Session in The Twelfth International World Wide Web Conference*, 2003.
- [17] I. Ruthven. Re-examining the potential effectiveness of interactive query expansion. In *Proceedings of the Twenty-Sixth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-2003)*, pages 213–220, 2003.
- [18] A. Spink, B. J. Jansen, and H. C. Ozmultu. Use of query reformulation and relevance feedback by Excite users. *Internet Research: Electronic Networking Applications and Policy*, 10(4):317–328, 2000.
- [19] E. Terra and C. L. A. Clarke. Scoring missing terms in information retrieval tasks. In *ACM Thirteenth Conference on Information and Knowledge Management (CIKM-2004)*, pages 50–58, 2004.