

Context Sensitive Stemming for Web Search

Fuchun Peng Nawaaz Ahmed Xin Li Yumao Lu
 Yahoo! Inc.
 701 First Avenue
 Sunnyvale, California 94089
 {fuchun, nawaaz, xinli, yumaol}@yahoo-inc.com

ABSTRACT

Traditionally, stemming has been applied to Information Retrieval tasks by transforming words in documents to their root form before indexing, and applying a similar transformation to query terms. Although it increases recall, this naive strategy does not work well for Web Search since it lowers precision and requires a significant amount of additional computation.

In this paper, we propose a context sensitive stemming method that addresses these two issues. Two unique properties make our approach feasible for Web Search. First, based on statistical language modeling, we perform context sensitive analysis on the query side. We accurately predict which of its morphological variants is useful to expand a query term with *before* submitting the query to the search engine. This dramatically reduces the number of bad expansions, which in turn reduces the cost of additional computation and improves the precision at the same time. Second, our approach performs a context sensitive document matching for those expanded variants. This conservative strategy serves as a safeguard against spurious stemming, and it turns out to be very important for improving precision. Using word pluralization handling as an example of our stemming approach, our experiments on a major Web search engine show that stemming only 29% of the query traffic, we can improve relevance as measured by average Discounted Cumulative Gain (DCG5) by 6.1% on these queries and 1.8% over all query traffic.

Categories and Subject Descriptors

H.3.3 [Information Systems]: Information Storage and Retrieval—*Query formulation*

General Terms

Algorithms, Experimentation

Keywords

Stemming, language modeling, Web search

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGIR'07, July 23–27, 2007, Amsterdam, The Netherlands.
 Copyright 2007 ACM 978-1-59593-597-7/07/0007 ...\$5.00.

1. INTRODUCTION

Web search has now become a major tool in our daily lives for information seeking. One of the important issues in Web search is that user queries are often not best formulated to get optimal results. For example, “running shoe” is a query that occurs frequently in query logs. However, the query “running shoes” is much more likely to give better search results than the original query because documents matching the intent of this query usually contain the words “running shoes”.

Correctly formulating a query requires the user to accurately predict which word form is used in the documents that best satisfy his or her information needs. This is difficult even for experienced users, and especially difficult for non-native speakers. One traditional solution is to use stemming [16, 18], the process of transforming inflected or derived words to their root form so that a search term will match and retrieve documents containing all forms of the term. Thus, the word “run” will match “running”, “ran”, “runs”, and “shoe” will match “shoes” and “shoeing”. Stemming can be done either on the terms in a document during indexing (and applying the same transformation to the query terms during query processing) or by expanding the query with the variants during query processing. Stemming during indexing allows very little flexibility during query processing, while stemming by query expansion allows handling each query differently, and hence is preferred.

Although traditional stemming increases recall by matching word variants [13], it can reduce precision by retrieving too many documents that have been incorrectly matched. When examining the results of applying stemming to a large number of queries, one usually finds that nearly equal numbers of queries are helped and hurt by the technique [6]. In addition, it reduces system performance because the search engine has to match all the word variants. As we will show in the experiments, this is true even if we simplify stemming to pluralization handling, which is the process of converting a word from its plural to singular form, or vice versa. Thus, one needs to be very cautious when using stemming in Web search engines.

One problem of traditional stemming is its blind transformation of all query terms, that is, it always performs the same transformation for the same query word without considering the context of the word. For example, the word “book” has four forms “book, books, booking, booked”, and “store” has four forms “store, stores, storing, stored”. For the query “book store”, expanding both words to all of their variants significantly increases computation cost and hurts

precision, since not all of the variants are useful for this query. Transforming “book store” to match “book stores” is fine, but matching “book storing” or “booking store” is not. A weighting method that gives variant words smaller weights alleviates the problems to a certain extent if the weights accurately reflect the importance of the variant in this particular query. However uniform weighting is not going to work and a query dependent weighting is still a challenging unsolved problem [20].

A second problem of traditional stemming is its blind matching of all occurrences in documents. For the query “book store”, a transformation that allows the variant “stores” to be matched will cause every occurrence of “stores” in the document to be treated equivalent to the query term “store”. Thus, a document containing the fragment “reading a book in coffee stores” will be matched, causing many wrong documents to be selected. Although we hope the ranking function can correctly handle these, with many more candidates to rank, the risk of making mistakes increases.

To alleviate these two problems, we propose a context sensitive stemming approach for Web search. Our solution consists of two context sensitive analysis, one on the query side and the other on the document side. On the query side, we propose a statistical language modeling based approach to predict which word variants are better forms than the original word for search purpose and expanding the query with only those forms. On the document side, we propose a conservative context sensitive matching for the transformed word variants, only matching document occurrences in the context of other terms in the query. Our model is simple yet effective and efficient, making it feasible to be used in real commercial Web search engines.

We use pluralization handling as a running example for our stemming approach. The motivation for using pluralization handling as an example is to show that even such simple stemming, if handled correctly, can give significant benefits to search relevance. As far as we know, no previous research has systematically investigated the usage of pluralization in Web search. As we have to point out, the method we propose is not limited to pluralization handling, it is a general stemming technique, and can also be applied to general query expansion. Experiments on general stemming yield additional significant improvements over pluralization handling for long queries, although details will not be reported in this paper.

In the rest of the paper, we first present the related work and distinguish our method from previous work in Section 2. We describe the details of the context sensitive stemming approach in Section 3. We then perform extensive experiments on a major Web search engine to support our claims in Section 4, followed by discussions in Section 5. Finally, we conclude the paper in Section 6.

2. RELATED WORK

Stemming is a long studied technology. Many stemmers have been developed, such as the Lovins stemmer [16] and the Porter stemmer [18]. The Porter stemmer is widely used due to its simplicity and effectiveness in many applications. However, the Porter stemming makes many mistakes because its simple rules cannot fully describe English morphology. Corpus analysis is used to improve Porter stemmer [26] by creating equivalence classes for words that are morphologically similar and occur in similar context as measured by

expected mutual information [23]. We use a similar corpus based approach for stemming by computing the similarity between two words based on their distributional context features which can be more than just adjacent words [15], and then only keep the morphologically similar words as candidates.

Using stemming in information retrieval is also a well known technique [8, 10]. However, the effectiveness of stemming for English query systems was previously reported to be rather limited. Lennon et al. [17] compared the Lovins and Porter algorithms and found little improvement in retrieval performance. Later, Harman [9] compares three general stemming techniques in text retrieval experiments including pluralization handling (called S stemmer in the paper). They also proposed selective stemming based on query length and term importance, but no positive results were reported. On the other hand, Krovetz [14] performed comparisons over small numbers of documents (from 400 to 12k) and showed dramatic precision improvement (up to 45%). However, due to the limited number of tested queries (less than 100) and the small size of the collection, the results are hard to generalize to Web search. These mixed results, mostly failures, led early IR researchers to deem stemming irrelevant in general for English [4], although recent research has shown stemming has greater benefits for retrieval in other languages [2]. We suspect the previous failures were mainly due to the two problems we mentioned in the introduction. Blind stemming, or a simple query length based selective stemming as used in [9] is not enough. Stemming has to be decided on case by case basis, not only at the query level but also at the document level. As we will show, if handled correctly, significant improvement can be achieved.

A more general problem related to stemming is query reformulation [3, 12] and query expansion which expands words not only with word variants [7, 22, 24, 25]. To decide which expanded words to use, people often use pseudo-relevance feedback techniques that send the original query to a search engine and retrieve the top documents, extract relevant words from these top documents as additional query words, and resubmit the expanded query again [21]. This normally requires sending a query multiple times to search engine and it is not cost effective for processing the huge amount of queries involved in Web search. In addition, query expansion, including query reformulation [3, 12], has a high risk of changing the user intent (called *query drift*). Since the expanded words may have different meanings, adding them to the query could potentially change the intent of the original query. Thus query expansion based on pseudo-relevance and query reformulation can provide suggestions to users for interactive refinement but can hardly be directly used for Web search. On the other hand, stemming is much more conservative since most of the time, stemming preserves the original search intent. While most work on query expansion focuses on recall enhancement, our work focuses on increasing both recall and precision. The increase on recall is obvious. With quality stemming, good documents which were not selected before stemming will be pushed up and those low quality documents will be degraded.

On selective query expansion, Cronen-Townsend et al. [6] proposed a method for selective query expansion based on comparing the Kullback-Leibler divergence of the results from the unexpanded query and the results from the expanded query. This is similar to the relevance feedback in

the sense that it requires multiple passes retrieval. If a word can be expanded into several words, it requires running this process multiple times to decide which expanded word is useful. It is expensive to deploy this in production Web search engines. Our method predicts the quality of expansion based on offline information without sending the query to a search engine.

In summary, we propose a novel approach to attack an old, yet still important and challenging problem for Web search – stemming. Our approach is unique in that it performs predictive stemming on a per query basis without relevance feedback from the Web, using the context of the variants in documents to preserve precision. It’s simple, yet very efficient and effective, making real time stemming feasible for Web search. Our results will affirm researchers that stemming is indeed very important to large scale information retrieval.

3. CONTEXT SENSITIVE STEMMING

3.1 Overview

Our system has four components as illustrated in Figure 1: candidate generation, query segmentation and head word detection, context sensitive query stemming and context sensitive document matching. Candidate generation (component 1) is performed offline and generated candidates are stored in a dictionary. For an input query, we first segment query into concepts and detect the head word for each concept (component 2). We then use statistical language modeling to decide whether a particular variant is useful (component 3), and finally for the expanded variants, we perform context sensitive document matching (component 4). Below we discuss each of the components in more detail.

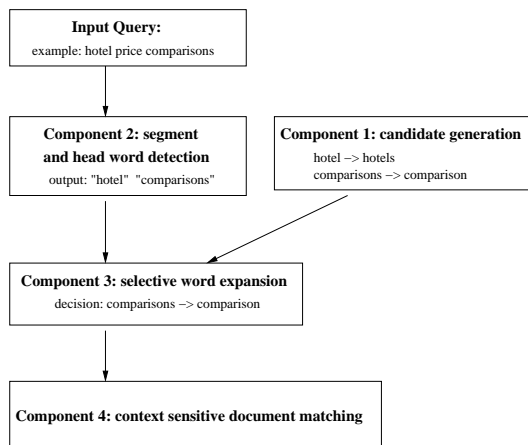


Figure 1: System Components

3.2 Expansion candidate generation

One of the ways to generate candidates is using the Porter stemmer [18]. The Porter stemmer simply uses morphological rules to convert a word to its base form. It has no knowledge of the semantic meaning of the words and sometimes makes serious mistakes, such as “executive” to “execution”, “news” to “new”, and “paste” to “past”. A more conservative way is based on using corpus analysis to improve the Porter stemmer results [26]. The corpus analysis we do is

based on word distributional similarity [15]. The rationale of using distributional word similarity is that true variants tend to be used in similar contexts. In the distributional word similarity calculation, each word is represented with a vector of features derived from the context of the word. We use the bigrams to the left and right of the word as its context features, by mining a huge Web corpus. The similarity between two words is the cosine similarity between the two corresponding feature vectors. The top 20 similar words to “develop” is shown in the following table.

rank	candidate	score	rank	candidate	score
0	develop	1	10	berts	0.119
1	developing	0.339	11	wads	0.116
2	developed	0.176	12	developer	0.107
3	incubator	0.160	13	promoting	0.100
4	develops	0.150	14	developmental	0.091
5	development	0.148	15	reengineering	0.090
6	tutoring	0.138	16	build	0.083
7	analyzing	0.128	17	construct	0.081
8	development	0.128	18	educational	0.081
9	automation	0.126	19	institute	0.077

Table 1: Top 20 most similar candidates to word “develop”. Column *score* is the similarity score.

To determine the stemming candidates, we apply a few Porter stemmer [18] morphological rules to the similarity list. After applying these rules, for the word “develop”, the stemming candidates are “developing, developed, develops, development, developement, developer, developmental”. For the pluralization handling purpose, only the candidate “develops” is retained.

One thing we note from observing the distributionally similar words is that they are closely related semantically. These words might serve as candidates for general query expansion, a topic we will investigate in the future.

3.3 Segmentation and headword identification

For long queries, it is quite important to detect the concepts in the query and the most important words for those concepts. We first break a query into segments, each segment representing a concept which normally is a noun phrase. For each of the noun phrases, we then detect the most important word which we call the head word. Segmentation is also used in document sensitive matching (section 3.5) to enforce proximity.

To break a query into segments, we have to define a criteria to measure the strength of the relation between words. One effective method is to use mutual information as an indicator on whether or not to split two words [19]. We use a log of 25M queries and collect the bigram and unigram frequencies from it. For every incoming query, we compute the mutual information of two adjacent words; if it passes a predefined threshold, we do not split the query between those two words and move on to next word. We continue this process until the mutual information between two words is below the threshold, then create a concept boundary here. Table 2 shows some examples of query segmentation.

The ideal way of finding the head word of a concept is to do syntactic parsing to determine the dependency structure of the query. Query parsing is more difficult than sentence

[running shoe]
[best] [new york] [medical schools]
[pictures] [of] [white house]
[cookies] [in] [san francisco]
[hotel] [price comparison]

Table 2: Query segmentation: a segment is bracketed.

parsing since many queries are not grammatical and are very short. Applying a parser trained on sentences from documents to queries will have poor performance. In our solution, we just use simple heuristics rules, and it works very well in practice for English. For an English noun phrase, the head word is typically the last nonstop word, unless the phrase is of a particular pattern, like “XYZ of/in/at/from UVW”. In such cases, the head word is typically the last nonstop word of XYZ.

3.4 Context sensitive word expansion

After detecting which words are the most important words to expand, we have to decide whether the expansions will be useful.

Our statistics show that about half of the queries can be transformed by pluralization via naive stemming. Among this half, about 25% of the queries improve relevance when transformed, the majority (about 50%) do not change their top 5 results, and the remaining 25% perform worse. Thus, it is extremely important to identify which queries should not be stemmed for the purpose of maximizing relevance improvement and minimizing stemming cost. In addition, for a query with multiple words that can be transformed, or a word with multiple variants, not all of the expansions are useful. Taking query “hotel price comparison” as an example, we decide that *hotel* and *price comparison* are two concepts. Head words “hotel” and “comparison” can be expanded to “hotels” and “comparisons”. Are both transformations useful?

To test whether an expansion is useful, we have to know whether the expanded query is likely to get more relevant documents from the Web, which can be quantified by the probability of the query occurring as a string on the Web. The more likely a query to occur on the Web, the more relevant documents this query is able to return. Now the whole problem becomes how to calculate the probability of query to occur on the Web.

Calculating the probability of string occurring in a corpus is a well known language modeling problem. The goal of language modeling is to predict the probability of naturally occurring word sequences, $s = w_1 w_2 \dots w_N$; or more simply, to put high probability on word sequences that actually occur (and low probability on word sequences that never occur). The simplest and most successful approach to language modeling is still based on the n -gram model. By the chain rule of probability one can write the probability of any word sequence as

$$\Pr(w_1 w_2 \dots w_N) = \prod_{i=1}^N \Pr(w_i | w_1 \dots w_{i-1}) \quad (1)$$

An n -gram model approximates this probability by assuming that the only words relevant to predicting $\Pr(w_i | w_1 \dots w_{i-1})$

are the previous $n - 1$ words; i.e.

$$\Pr(w_i | w_1 \dots w_{i-1}) = \Pr(w_i | w_{i-n+1} \dots w_{i-1})$$

A straightforward maximum likelihood estimate of n -gram probabilities from a corpus is given by the observed frequency of each of the patterns

$$\Pr(w_i | w_{i-n+1} \dots w_{i-1}) = \frac{\#(w_{i-n+1} \dots w_i)}{\#(w_{i-n+1} \dots w_{i-1})} \quad (2)$$

where $\#(\cdot)$ denotes the number of occurrences of a specified gram in the training corpus. Although one could attempt to use simple n -gram models to capture long range dependencies in language, attempting to do so directly immediately creates sparse data problems: Using grams of length up to n entails estimating the probability of W^n events, where W is the size of the word vocabulary. This quickly overwhelms modern computational and data resources for even modest choices of n (beyond 3 to 6). Also, because of the heavy tailed nature of language (i.e. Zipf’s law) one is likely to encounter novel n -grams that were never witnessed during training in any test corpus, and therefore some mechanism for assigning non-zero probability to novel n -grams is a central and unavoidable issue in statistical language modeling. One standard approach to smoothing probability estimates to cope with sparse data problems (and to cope with potentially missing n -grams) is to use some sort of back-off estimator.

$$\Pr(w_i | w_{i-n+1} \dots w_{i-1}) = \begin{cases} \hat{\Pr}(w_i | w_{i-n+1} \dots w_{i-1}), & \text{if } \#(w_{i-n+1} \dots w_i) > 0 \\ \beta(w_{i-n+1} \dots w_{i-1}) \times \Pr(w_i | w_{i-n+2} \dots w_{i-1}), & \text{otherwise} \end{cases} \quad (3)$$

where

$$\hat{\Pr}(w_i | w_{i-n+1} \dots w_{i-1}) = \frac{\text{discount} \#(w_{i-n+1} \dots w_i)}{\#(w_{i-n+1} \dots w_{i-1})} \quad (4)$$

is the discounted probability and $\beta(w_{i-n+1} \dots w_{i-1})$ is a normalization constant

$$\beta(w_{i-n+1} \dots w_{i-1}) = \frac{1 - \sum_{x \in (w_{i-n+1} \dots w_{i-1} x)} \hat{\Pr}(x | w_{i-n+1} \dots w_{i-1})}{1 - \sum_{x \in (w_{i-n+1} \dots w_{i-1} x)} \hat{\Pr}(x | w_{i-n+2} \dots w_{i-1})} \quad (5)$$

The discounted probability (4) can be computed with different smoothing techniques, including absolute smoothing, Good-Turing smoothing, linear smoothing, and Witten-Bell smoothing [5]. We used absolute smoothing in our experiments.

Since the likelihood of a string, $\Pr(w_1 w_2 \dots w_N)$, is a very small number and hard to interpret, we use entropy as defined below to score the string.

$$\text{Entropy} = -\frac{1}{N} \log_2 \Pr(w_1 w_2 \dots w_N) \quad (6)$$

Now getting back to the example of the query “hotel price comparisons”, there are four variants of this query, and the entropy of these four candidates are shown in Table 3. We can see that all alternatives are less likely than the input

query. It is therefore not useful to make an expansion for this query. On the other hand, if the input query is “hotel price comparisons” which is the second alternative in the table, then there is a better alternative than the input query, and it should therefore be expanded. To tolerate the variations in probability estimation, we relax the selection criterion to those query alternatives if their scores are within a certain distance (10% in our experiments) to the best score.

Query variations	Entropy
hotel price comparison	6.177
hotel price comparisons	6.597
hotels price comparison	6.937
hotels price comparisons	7.360

Table 3: Variations of query “hotel price comparison” ranked by entropy score, with the original query in bold face.

3.5 Context sensitive document matching

Even after we know which word variants are likely to be useful, we have to be conservative in document matching for the expanded variants. For the query “hotel price comparisons”, we decided that word “comparisons” is expanded to include “comparison”. However, not every occurrence of “comparison” in the document is of interest. A page which is about comparing customer service can contain all of the words *hotel price comparisons comparison*. This page is not a good page for the query.

If we accept matches of every occurrence of “comparison”, it will hurt retrieval precision and this is one of the main reasons why most stemming approaches do not work well for information retrieval. To address this problem, we have a proximity constraint that considers the context around the expanded variant in the document. A variant match is considered valid only if the variant occurs in the same context as the original word does. The context is the left or the right non-stop segments¹ of the original word. Taking the same query as an example, the context of “comparisons” is “price”. The expanded word “comparison” is only valid if it is in the same context of “comparisons”, which is after the word “price”. Thus, we should only match those occurrences of “comparison” in the document if they occur after the word “price”. Considering the fact that queries and documents may not represent the intent in exactly the same way, we relax this proximity constraint to allow variant occurrences within a window of some fixed size. If the expanded word “comparison” occurs within the context of “price” within a window, it is considered valid. The smaller the window size is, the more restrictive the matching. We use a window size of 4, which typically captures contexts that include the containing and adjacent noun phrases.

4. EXPERIMENTAL EVALUATION

4.1 Evaluation metrics

We will measure both relevance improvement and the stemming cost required to achieve the relevance.

¹a context segment can not be a single stop word.

4.1.1 Relevance measurement

We use a variant of the average Discounted Cumulative Gain (DCG), a recently popularized scheme to measure search engine relevance [1, 11]. Given a query and a ranked list of K documents (K is set to 5 in our experiments), the $DCG(K)$ score for this query is calculated as follows:

$$DCG(K) = \sum_{k=1}^K \frac{g_k}{\log_2(1+k)}. \quad (7)$$

where g_k is the weight for the document at rank k . Higher degree of relevance corresponds to a higher weight. A page is graded into one of the five scales: Perfect, Excellent, Good, Fair, Bad, with corresponding weights. We use d_{cg} to represent the average $DCG(5)$ over a set of test queries.

4.1.2 Stemming cost

Another metric is to measure the additional cost incurred by stemming. Given the same level of relevance improvement, we prefer a stemming method that has less additional cost. We measure this by the percentage of queries that are actually stemmed, over all the queries that could possibly be stemmed.

4.2 Data preparation

We randomly sample 870 queries from a three month query log, with 290 from each month. Among all these 870 queries, we remove all misspelled queries since misspelled queries are not of interest to stemming. We also remove all one word queries since stemming one word queries without context has a high risk of changing query intent, especially for short words. In the end, we have 529 correctly spelled queries with at least 2 words.

4.3 Naive stemming for Web search

Before explaining the experiments and results in detail, we’d like to describe the traditional way of using stemming for Web search, referred as the *naive model*. This is to treat every word variant equivalent for all possible words in the query. The query “book store” will be transformed into “(book OR books)(store OR stores)” when limiting stemming to pluralization handling only, where *OR* is an operator that denotes the equivalence of the left and right arguments.

4.4 Experimental setup

The *baseline model* is the model without stemming. We first run the *naive model* to see how well it performs over the baseline. Then we improve the naive stemming model by document sensitive matching, referred as *document sensitive matching model*. This model makes the same stemming as the naive model on the query side, but performs conservative matching on the document side using the strategy described in section 3.5. The *naive model* and *document sensitive matching model* stem the most queries. Out of the 529 queries, there are 408 queries that they stem, corresponding to 46.7% query traffic (out of a total of 870). We then further improve the *document sensitive matching model* from the query side with selective word stemming based on statistical language modeling (section 3.4), referred as *selective stemming model*. Based on language modeling prediction, this model stems only a subset of the 408 queries stemmed by the *document sensitive matching model*. We experiment with unigram language model and bigram lan-

guage model. Since we only care how much we can improve the naive model, we will only use these 408 queries (all the queries that are affected by the naive stemming model) in the experiments.

To get a sense of how these models perform, we also have an *oracle model* that gives the upper-bound performance a stemmer can achieve on this data. The oracle model only expands a word if the stemming will give better results.

To analyze the pluralization handling influence on different query categories, we divide queries into short queries and long queries. Among the 408 queries stemmed by the naive model, there are 272 short queries with 2 or 3 words, and 136 long queries with at least 4 words.

4.5 Results

We summarize the overall results in Table 4, and present the results on short queries and long queries separately in Table 5. Each row in Table 4 is a stemming strategy described in section 4.4. The first column is the name of the strategy. The second column is the number of queries affected by this strategy; this column measures the stemming cost, and the numbers should be low for the same level of *dcg*. The third column is the average *dcg* score over all tested queries in this category (including the ones that were not stemmed by the strategy). The fourth column is the relative improvement over the baseline, and the last column is the *p*-value of Wilcoxon significance test.

There are several observations about the results. We can see the naively stemming only obtains a statistically *insignificant* improvement of **1.5%**. Looking at Table 5, it gives an improvement of **2.7%** on short queries. However, it also hurts long queries by **-2.4%**. Overall, the improvement is canceled out. The reason that it improves short queries is that most short queries only have one word that can be stemmed. Thus, blindly pluralizing short queries is relatively safe. However for long queries, most queries can have multiple words that can be pluralized. Expanding all of them without selection will significantly hurt precision.

Document context sensitive stemming gives a *significant* lift to the performance, from **2.7%** to **4.2%** for short queries and from **-2.4%** to **-1.6%** for long queries, with an overall lift from **1.5%** to **2.8%**. The improvement comes from the conservative context sensitive document matching. An expanded word is valid only if it occurs within the context of original query in the document. This reduces many spurious matches. However, we still notice that for long queries, context sensitive stemming is not able to improve performance because it still selects too many documents and gives the ranking function a hard problem. While the chosen window size of 4 works the best amongst all the choices, it still allows spurious matches. It is possible that the window size needs to be chosen on a per query basis to ensure tighter proximity constraints for different types of noun phrases.

Selective word pluralization further helps resolving the problem faced by document context sensitive stemming. It does not stem every word that places all the burden on the ranking algorithm, but tries to eliminate unnecessary stemming in the first place. By predicting which word variants are going to be useful, we can dramatically reduce the number of stemmed words, thus improving both the recall and the precision. With the unigram language model, we can reduce the stemming cost by 26.7% (from 408/408 to 300/408) and lift the overall *dcg* improvement from **2.8%** to **3.4%**. In

particular, it gives significant improvements on long queries. The *dcg* gain is turned from negative to positive, from **-1.6%** to **1.1%**. This confirms our hypothesis that reducing unnecessary word expansion leads to precision improvement. For short queries too, we observe both *dcg* improvement and stemming cost reduction with the unigram language model.

The advantages of predictive word expansion with a language model is further boosted with a better bigram language model. The overall *dcg* gain is lifted from **3.4%** to **3.9%**, and stemming cost is dramatically reduced from 408/408 to 250/408, corresponding to only **29%** of query traffic (250 out of 870) and an overall **1.8%** *dcg* improvement overall all query traffic. For short queries, bigram language model improves the *dcg* gain from **4.4%** to **4.7%**, and reduces stemming cost from 272/272 to 150/272. For long queries, bigram language model improves *dcg* gain from **1.1%** to **2.5%**, and reduces stemming cost from 136/136 to 100/136. We observe that the bigram language model gives a larger lift for long queries. This is because the uncertainty in long queries is larger and a more powerful language model is needed. We hypothesize that a trigram language model would give a further lift for long queries and leave this for future investigation.

Considering the tight upper-bound² on the improvement to be gained from pluralization handling (via the oracle model), the current performance on short queries is very satisfying. For short queries, the *dcg* gain upper-bound is 6.3% for perfect pluralization handling, our current gain is **4.7%** with a bigram language model. For long queries, the *dcg* gain upper-bound is 4.6% for perfect pluralization handling, our current gain is **2.5%** with a bigram language model. We may gain additional benefit with a more powerful language model for long queries. However, the difficulties of long queries come from many other aspects including the proximity and the segmentation problem. These problems have to be addressed separately. Looking at the the upper-bound of overhead reduction for oracle stemming, 75% (308/408) of the naive stemmings are wasteful. We currently capture about half of them. Further reduction of the overhead requires sacrificing the *dcg* gain.

Now we can compare the stemming strategies from a different aspect. Instead of looking at the influence over all queries as we described above, Table 6 summarizes the *dcg* improvements over the affected queries only. We can see that the number of affected queries decreases as the stemming strategy becomes more accurate (*dcg* improvement). For the bigram language model, over the 250/408 stemmed queries, the *dcg* improvement is **6.1%**. An interesting observation is the average *dcg* decreases with a better model, which indicates a better stemming strategy stems more difficult queries (low *dcg* queries).

5. DISCUSSIONS

5.1 Language models from query vs. from Web

As we mentioned in Section 1, we are trying to predict the probability of a string occurring on the Web. The language model should describe the occurrence of the string on the Web. However, the query log is also a good resource.

²Note that this upperbound is for pluralization handling only, not for general stemming. General stemming gives a 8% upperbound, which is quite substantial in terms of our metrics.

	Affected Queries	<i>dcg</i>	<i>dcg</i> Improvement	<i>p</i> -value
baseline	0/408	7.102	N/A	N/A
naive model	408/408	7.206	1.5%	0.22
document context sensitive model	408/408	7.302	2.8%	0.014
selective model: unigram LM	300/408	7.321	3.4%	0.001
selective model: bigram LM	250/408	7.381	3.9%	0.001
oracle model	100/408	7.519	5.9%	0.001

Table 4: Results comparison of different stemming strategies over all queries affected by naive stemming

Short Query Results			
	Affected Queries	<i>dcg</i> Improvement	<i>p</i> -value
baseline	0/272	N/A	N/A
naive model	272/272	2.7%	0.48
document context sensitive model	272/272	4.2%	0.002
selective model: unigram LM	185/272	4.4%	0.001
selective model: bigram LM	150/272	4.7%	0.001
oracle model	71/272	6.3%	0.001
Long Query Results			
	Affected Queries	<i>dcg</i> Improvement	<i>p</i> -value
baseline	0/136	N/A	N/A
naive model	136/136	-2.4%	0.25
document context sensitive model	136/136	-1.6%	0.27
selective model: unigram LM	115/136	1.1%	0.001
selective model: bigram LM	100/136	2.5%	0.001
oracle model	29/136	4.6%	0.001

Table 5: Results comparison of different stemming strategies overall short queries and long queries

Users reformulate a query using many different variants to get good results.

To test the hypothesis that we can learn reliable transformation probabilities from the query log, we trained a language model from the same query top 25M queries as used to learn segmentation, and use that for prediction. We observed a slight performance decrease compared to the model trained on Web frequencies. In particular, the performance for unigram LM was not affected, but the *dcg* gain for bigram LM changed from 4.7% to 4.5% for short queries. Thus, the query log can serve as a good approximation of the Web frequencies.

5.2 How linguistics helps

Some linguistic knowledge is useful in stemming. For the pluralization handling case, pluralization and de-pluralization is not symmetric. A plural word used in a query indicates a special intent. For example, the query “new york hotels” is looking for a list of hotels in new york, not the specific “new york hotel” which might be a hotel located in California. A simple equivalence of “hotel” to “hotels” might boost a particular page about “new york hotel” to top rank. To capture this intent, we have to make sure the document is a general page about hotels in new york. We do this by requiring that the plural word “hotels” appears in the document. On the other hand, converting a singular word to plural is safer since a general purpose page normally contains specific information. We observed a slight overall *dcg* decrease, although not statistically significant, for document context sensitive stemming if we do not consider this asymmetric property.

5.3 Error analysis

One type of mistakes we noticed, though rare but seriously hurting relevance, is the search intent change after stemming. Generally speaking, pluralization or depluralization keeps the original intent. However, the intent could change in a few cases. For one example of such a query, “job at apple”, we pluralize “job” to “jobs”. This stemming makes the original query ambiguous. The query “job OR jobs at apple” has two intents. One is the employment opportunities at apple, and another is a person working at Apple, Steve Jobs, who is the CEO and co-founder of the company. Thus, the results after query stemming returns “Steve Jobs” as one of the results in top 5. One solution is performing results set based analysis to check if the intent is changed. This is similar to relevance feedback and requires second phase ranking.

A second type of mistakes is the entity/concept recognition problem. These include two kinds. One is that the stemmed word variant now matches part of an entity or concept. For example, query “cookies in san francisco” is pluralized to “cookies OR cookie in san francisco”. The results will match “cookie jar in san francisco”. Although “cookie” still means the same thing as “cookies”, “cookie jar” is a different concept. Another kind is the unstemmed word matches an entity or concept because of the stemming of the other words. For example, “quote ICE” is pluralized to “quote OR quotes ICE”. The original intent for this query is searching for stock quote for ticker ICE. However, we noticed that among the top results, one of the results is “Food quotes: Ice cream”. This is matched because of

	Affected Queries	old <i>dcg</i>	new <i>dcg</i>	<i>dcg</i> Improvement
naive model	408/408	7.102	7.206	1.5%
document context sensitive model	408/408	7.102	7.302	2.8%
selective model: unigram LM	300/408	5.904	6.187	4.8%
selective model: bigram LM	250/408	5.551	5.891	6.1%

Table 6: Results comparison over the stemmed queries only: column old/new *dcg* is the *dcg* score over the affected queries before/after applying stemming

the pluralized word “quotes”. The unchanged word “ICE” matches part of the noun phrase “ice cream” here. To solve this kind of problem, we have to analyze the documents and recognize “cookie jar” and “ice cream” as concepts instead of two independent words.

A third type of mistakes occurs in long queries. For the query “bar code reader software”, two words are pluralized. “code” to “codes” and “reader” to “readers”. In fact, “bar code reader” in the original query is a strong concept and the internal words should not be changed. This is the segmentation and entity and noun phrase detection problem in queries, which we actively are attacking. For long queries, we should correctly identify the concepts in the query, and boost the proximity for the words within a concept.

6. CONCLUSIONS AND FUTURE WORK

We have presented a simple yet elegant way of stemming for Web search. It improves naive stemming in two aspects: selective word expansion on the query side and conservative word occurrence matching on the document side. Using pluralization handling as an example, experiments on a major Web search engine data show it significantly improves the Web relevance and reduces the stemming cost. It also significantly improves Web click through rate (details not reported in the paper).

For the future work, we are investigating the problems we identified in the error analysis section. These include: entity and noun phrase matching mistakes, and improved segmentation.

7. REFERENCES

- [1] E. Agichtein, E. Brill, and S. T. Dumais. Improving Web Search Ranking by Incorporating User Behavior Information. In *SIGIR*, 2006.
- [2] E. Airio. Word Normalization and Decomposition in Mono- and Bilingual IR. *Information Retrieval*, 9:249–271, 2006.
- [3] P. Anick. Using Terminological Feedback for Web Search Refinement: a Log-based Study. In *SIGIR*, 2003.
- [4] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press/Addison Wesley, 1999.
- [5] S. Chen and J. Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. Technical Report TR-10-98, Harvard University, 1998.
- [6] S. Cronen-Townsend, Y. Zhou, and B. Croft. A Framework for Selective Query Expansion. In *CIKM*, 2004.
- [7] H. Fang and C. Zhai. Semantic Term Matching in Axiomatic Approaches to Information Retrieval. In *SIGIR*, 2006.
- [8] W. B. Frakes. Term Conflation for Information Retrieval. In C. J. Rijsbergen, editor, *Research and Development in Information Retrieval*, pages 383–389. Cambridge University Press, 1984.
- [9] D. Harman. How Effective is Suffixing? *JASIS*, 42(1):7–15, 1991.
- [10] D. Hull. Stemming Algorithms - A Case Study for Detailed Evaluation. *JASIS*, 47(1):70–84, 1996.
- [11] K. Jarvelin and J. Kekalainen. Cumulated Gain-Based Evaluation Evaluation of IR Techniques. *ACM TOIS*, 20:422–446, 2002.
- [12] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating Query Substitutions. In *WWW*, 2006.
- [13] W. Kraaij and R. Pohlmann. Viewing Stemming as Recall Enhancement. In *SIGIR*, 1996.
- [14] R. Krovetz. Viewing Morphology as an Inference Process. In *SIGIR*, 1993.
- [15] D. Lin. Automatic Retrieval and Clustering of Similar Words. In *COLING-ACL*, 1998.
- [16] J. B. Lovins. Development of a Stemming Algorithm. *Mechanical Translation and Computational Linguistics*, II:22–31, 1968.
- [17] M. Lennon and D. Peirce and B. Tarry and P. Willett. An Evaluation of Some Conflation Algorithms for Information Retrieval. *Journal of Information Science*, 3:177–188, 1981.
- [18] M. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.
- [19] K. M. Risvik, T. Mikolajewski, and P. Boros. Query Segmentation for Web Search. In *WWW*, 2003.
- [20] S. E. Robertson. On Term Selection for Query Expansion. *Journal of Documentation*, 46(4):359–364, 1990.
- [21] G. Salton and C. Buckley. Improving Retrieval Performance by Relevance Feedback. *JASIS*, 41(4):288–297, 1999.
- [22] R. Sun, C.-H. Ong, and T.-S. Chua. Mining Dependency Relations for Query Expansion in Passage Retrieval. In *SIGIR*, 2006.
- [23] C. Van Rijsbergen. *Information Retrieval*. Butterworths, second version, 1979.
- [24] B. Vélez, R. Weiss, M. A. Sheldon, and D. K. Gifford. Fast and Effective Query Refinement. In *SIGIR*, 1997.
- [25] J. Xu and B. Croft. Query Expansion using Local and Global Document Analysis. In *SIGIR*, 1996.
- [26] J. Xu and B. Croft. Corpus-based Stemming using Cooccurrence of Word Variants. *ACM TOIS*, 16(1):61–81, 1998.