Implicit Queries for Email

Joshua Goodman
Machine Learning and Applied Statistics Group
Microsoft Research
One Microsoft Way
Redmond, WA 98052
joshuago@microsoft.com

Vitor R. Carvalho
Language Technologies Institute
School of Computer Science
Carnegie Melon University
Pittsburgh, PA
vitor@cs.cmu.edu

Abstract

Implicit query systems examine a document and automatically conduct searches for the most relevant information. In this paper, we offer three contributions to implicit query research. First, we show how to use query logs from a search engine: by constraining results to commonly issued queries, we can get dramatic improvements. Second, we describe a method for optimizing parameters for an implicit query system, by using logistic regression training. The method is designed to estimate the probability that any particular suggested query is a good one. Third, we show which features beyond standard TF-IDF features are most helpful in our logistic regression model: query frequency information, capitalization information, subject line information, and message length information. Using the optimization method and the additional features, we are able to produce a system with up to 6 times better results on top-1 score than a simple TF-IDF system.

1 Introduction

In this paper, we examine implicit query systems for email: automatically finding good words or phrases in an email message to send to a search engine. We will show three main results related to these efforts. First, we show how to use the query logs from a large search engine. By restricting queries to those commonly found in the logs, we can dramatically improve our results. Second, we show that email-specific information, specifically giving large weight to subject line information, can also lead to improvements. Third, we show that we can train a system to estimate probabilities that also has good performance on Top-1 and Top-10 scores.

Email is the number one activity people pursue online (Madden and Rainie, 2003), with Internet Search a very close second (*ibid.*) Given people's interests, it makes sense to combine these two technologies as much as possible. In addition, search is very lucrative: Google, for instance, appears to earn over a billion dollars a year from search (Google SEC Filing, 2004), besides other revenues. Anything we can do to encourage users to search is a win for both users and software producers.

There are many ways to make it easier for users to search while reading email, such as including easily accessible search boxes, or allowing users to select text and then right-click select search as an option. Easier still are single click solutions, such as converting likely words and phrases in a message to clickable links to search, or including such words and phrases in a sidebar. The easiest of all for users is to have already executed the query and return the result, as has been done by Google's Gmail system. These techniques in which likely queries are found and then turned into single-click or pre-retrieved solutions are known as implicit query systems.

A large portion of the previous literature on implicit query has focused either on the user interface, or on a related problem, also known as implicit query, of finding related documents in a database. For instance, when viewing an email message, you might be shown other related messages. These other messages can be found by, e.g., examining the similarity between the current message and all messages previously received. In this paper, we will be concerned with the problem of finding the best queries to retrieve internet search results.

We apply this research to the problem of finding good queries for email messages, but we expect variations could be used to find good queries for other document types, e.g. instant messaging, chat, or web pages. In addition, the area of contextual advertising is starting to receive substantial attention, and presumably these techniques could also be applied to the area of finding keywords for contextual ads for email, or any of the other document types. Much of the research in this paper focused on email-specific features. Disappointingly, in most cases, with the exception of subject line information, these features did not seem helpful. On the other hand, this is good news, in the sense that it means that our results can probably be applied to other document types.

In addition to finding the best queries, we also want to know how likely it is that each query is a relevant one. This gives us useful information for real applications. For instance, we may find that the most probable phrase found has only a .001 chance of being relevant. In this case, we might choose to show no results at all, so as not to distract the user, or an advertisement, or a different kind of relevant information, such as a history of other messages sent by the sender. We might also use the relative probabilities of various queries. For instance, if the top ranked query has probablity .2 of being relevant, and the second ranked query has probability .19, we might show two search results for each query, instead of four results for the first one.

Of course, we could have similar rules for, e.g., absolute or relative TF-IDF scores. However, the use of probabilities seems simpler and easier to make decisions about. In addition, it may be more robust to changes in the system. For example, changing from unstemmed to stemmed words might change all of the important thresholds for a TF-IDF system, while a system directly trained to estimate probabilities, when retrained with the stemmed words, should mostly automatically learn new parameter values that compensate for any such changes.

Because of our goal to return absolute probabilities of relevance, we also experimented with the use of information not typically used in an implicit query system, what we will refer to as global features of a message. For instance, we will use features such as whether the subject line starts with "Re:" or "Fwd:". Messages that are replies tend to be of more interest to a user, and thus there tends to be a higher probability that they will contain queries relevant to a user. In our system, the use of these features does not affect the relative ranking of possible query words, but it does affect their absolute probability, making the results more useful in real applications.

2 Method

2.1 Learning Algorithm

Our desire to get actual probabilities of relevance impacts our choice of model. Traditional choices for implicit query systems, such as TF-IDF, do not return quantities that are directly interpretable as probabilities. We chose to focus on logistic regression models, which, as we will show, can include TF-IDF and other traditional choices as special cases, but also naturally return probabilities. Logistic regression models are also called maximum entropy models in some communities, and are equivalent to a certain kind of single layer neural network. In particular, logistic regression models are of the form

$$P(y \mid \overline{x}) = \frac{\exp(\overline{w} \cdot \overline{x})}{1 + \exp(\overline{w} \cdot \overline{x})}$$

In this equation, y is the variable being predicted (in this case, y takes the values 0 or 1, with 1 meaning that a particular word or phrase is a good query for a particular message.) $\overline{x}$ is a vector of numbers representing the features of a particular word or message in an email message. For instance, our features might include the number of times that the word or phrase occurs in the subject; the number of times it occurs anywhere in the body; and 0 or 1 representing whether the word or phrase is capitalized. Finally, $\overline{w}$ represents a set of *weights*. These weights indicate the relative weights for each feature for each word or phrase; if subject words are twice as important as body words, w_1 might have twice the value of w_2. Typically, we learn these weights using training data, in this case, a corpus of messages for which we have hand labeled which words or phrases are relevant. Essentially, for every word or phrase (up to length 5) in each message, we have a training example, with value y=1 if the word was labeled as relevant, 0 otherwise. The vast majority of words are labeled as irrelevant. We then use a learning algorithm that maximizes the probability of the training data, assigning as large a probability as possible to those words or phrases that were relevant, and as small as possible to those that were not.

The training algorithm we use is Sequential Conditional Generalized Iterative Scaling (SCGIS) (Goodman, 2002), although because logistic regression models have a global optimum, the choice of learning algorithm is typically of little importance. Once the data has been extracted for the learning algorithm, the actual learning takes only a few seconds. One very important observation is that the algorithm must be run past the usual criteria for convergence: we use 100 iterations instead of our usual 10. The algorithm is fast, so even 100 iterations is not problematic.

Logistic regression models, like many other kinds of machine learning, have a tendency to overfit the training data. We use a Gaussian Prior (Chen et al., 1999) with variance of 1 to prevent overfitting.

2.2 Features

We experimented with a large number of features that we thought might be helpful. These included:

- **S: InSubject** How many times the word or phrase occurs in the subject.
- **A: Anywhere** How many times the word or phrase occurs anywhere in the body or subject of the message.

- **B: Body** How many times the word or phrase occurs in the body of the message.
- **G: AtBeginning** How many times the word or phrase occurs at the beginning of the message (in the first 50 characters of the body.) 50 was selected semi-arbitrarily.
- **H: NearBeginning** How many times the word or phrase occurs near the beginning of the message (in the first 300 characters of the body.) 300 was selected semi-arbitrarily.
- **R: BodyNotReply** How many times the word or phrase occurs in the body of the message, not including replied-to sections.
- **S_{DF}, A_{DF}, B_{DF}, G_{DF}, H_{DF}, R_{DF}: DFInSubject, DFAnywhere, DFBody, DFAtBeginning, DFNearBeginning, DFBodyNotReply** These are the document frequencies for a given word or phrase, i.e. the total number of occurrences of that word or phrase occurred in in the corpus.
- **Q: QueryFrequency** This is the number of times the word or phrase occurred in a corpus of MSN Search queries, as described below.
- **L: LogQueryFrequency** This is log(*QueryFrequency* + 1). See below.
- **P: PhraseLength** How long is the query phrase, in words.
- **C: Capitalized** Every time the word or phrase occurs, is the first letter of every word in the phrase capitalized? (Takes value 0 or 1 only)
- **T: SentenceBeginning** Does the word or phrase occur at the beginning of a sentence, somewhere in the message? (0 or 1 only)
- **M: MessageLength** This is the length of the message. We hypothesize that the longer a message is, the more difficult it is to find the "good" keywords in the message, separating them from bad ones. Also, see below about length normalization.
- **F, E: HasFWSubject, HasRESubject** Whether the subject line starts with FW or RE respectively (0 or 1 only).

For a feature occurring f_i times in a message, we actually use the value $x_i = \log(f_i+1)$ instead of $x_i=f_i$. This has commonly been found to be useful in areas such as information retrieval and text classification. (Because of the model form, the base of the logarithm does not matter.)

Many machine learning and information retrieval techniques use length normalization. Rather than explicitly perform length normalization, we allow the system to weight the length if desired, which provides an excellent approximation. For instance, imagine the ideal weights are

$$3 \times \log((\mathit{insubject}+1)/(\mathit{messagelength}+1)) \\ + 2 \times \log((\mathit{body}+1)/(\mathit{messagelength}+1))$$

where *insubject* and *body* represent the number of times a particular word or phrase occurs in the subject or body respectively. Since we do not provide length normalized features, it may not be obvious that the system can learn this model. But in fact, by not normalizing, and providing message length as a learnable feature, it can learn

$$3 \times \log(\mathit{insubject}+1) \\ + 2 \times \log(\mathit{body}+1) \\ - 5 \times \log(\mathit{message\text{-}length}+1)$$

which is equivalent. This provides our learning system flexibility, essentially allowing it to learn which features to normalize and which not to. In the end, because of the model form, it does not matter exactly which features are length normalized: the normalizations are implicitly summed.

Similary, we do not explicity use the formula TermFrequency/DocumentFrequency. Instead, we allow the system to learn the ideal weightings. We provide both TF and DF scores for each field, and allow the system to learn the appropriate weights; they may be, e.g. 1 and -1, yielding the conventional TF/DF = TF × IDF measure; or they may lead to other values.

We also used a corpus of the 7.5 million most common english language queries from MSN Search. In most of our experiments, we limited ourselves to returning queries that are in those 7.5 million. This made many experiments much more efficient: rather than considering all words and phrases in each message, we only considered those that were moderately likely to be queried by users. Among other things, since we extract both positive and negative examples of words or phrases as training data, it drastically reduced the number of negative training examples. It also, as we will show, dramatically improves the accuracy of the results, since it prevents us from returning words or phrases that would never be queried by real users.

We used the query frequency as an additional input to our system. Our reasoning was that perhaps if users queried for words or phrases more, they were more likely to be good phrases for implicit queries. This is the QueryFrequency feature described above. We thought, however, that QueryFrequency might be too blunt an instrument, and that distinguishing very high frequency queries from very low frequency queries might be more important. We thus also included a feature LogQueryFrequency, which was set to log(*QueryFrequency* + 1). As we've mentioned, since we add 1 and take the log of features before passing them to the logistic regression model, the actual values for x in the logistic regression model were not QueryFrequency and log(*QueryFrequency* + 1), but were instead log(*QueryFrequency*+1) and log(log(*QueryFrequency*+1)+1). Because of this transformation, for the LogQueryFrequency feature, there is very little difference between, say, a query occurring 100,000 times and one occurring 200,000 times; effectively, all very high frequency queries receive a similar value. On the other hand, there is a relatively large difference between features occurring say 20 times versus 40 times. This means that this feature is mostly sensitive to changes in frequency at the low end. By combining the two features, the learning system can control weights separately for high frequency and low frequency queries.

3 Experimental Results

3.1 Data and Evaluation Criteria

We initially planned to use the Enron corpus for these experiments, so that we could make our labeling of the corpus publicly available. However, a brief inspection of the Enron corpus showed that most of the messages were from one Enron user to another, and about projects specific to Enron: there would be little of interest to search for on the internet. A possible area of future research would be implicit query of email messages for intranet use, or for a desktop search application; see related work by Dumais et al. (2004), for an example of this.

For our data set, we used a corpus collected from 20 Hotmail volunteers from the general population (not Microsoft employees.) The volunteers had been directed to save all of their email, and to then hand-classify it into three sets: “spam,” “subs”, and “wanted.” Some words, such as users’ names, local words (e.g. about Redmond, WA and environs, where the users lived), and some company names had been removed from the corpus. The subs set was for mail that was not spam, but that the user did not particularly care about; this was primarily commercial mail for which the users had opted in, or for which they had a pre-existing business relationship, and had not opted out, as well as some mailing list mail. We focused on the “wanted” set, reasoning that this was the kind of mail where implicit query would be most useful. Of the wanted messages, we hand labeled approximately 1143 messages. For each message, our 6 annotators were instructed roughly as follows:

> These are mail messages from real Hotmail users. Imagine that you were the recipient of each message. If your email program were to automatically perform a query to a search engine like MSN Search or Google, for you, what words or phrases would you want the engine to search for? In some messages, there may be no words worth searching for. In others, there may be several. When possible, the words or phrases should actually occur in the messages you annotate.

For experimental purposes we eliminated any words or phrases that annotators entered that did not appear in a message. Since all methods we consider in this paper only return words and phrases actually present in the message, this affects all our methods equally.

All experiments were done with 10-way cross validation. The training data was split into ten pieces; we used nine for training any parameters that needed to be optimized, and one for test; this was repeated on each of the ten sets, and the results were averaged. In our experiments, no preprocessing was done: attachments, multiple MIME parts and HTML were all included.

We will present three kinds of results. The first type is the average **entropy** of test data. For each test item, we compute the $-\log_2 P(y|\bar{x})$ – the log of the probability of the training data. A test item is a word or a phrase (up to length 5) in a message. The entropy will be large if the model does a bad job of predicting which words and phrases are good matches (according to our labelers), and small if the model does a good job. The ideal entropy is 0, which happens when the model is perfect at predicting which words and phrases are good matches. Entropy is a useful measure if we care about getting probabilities right, e.g. if we might display no matches to a user if none are sufficiently likely. Note that logistic regression models are trained to minimize entropy.

We will also compute the **Top-1** score. For the Top-1 score, we compute for each message the best word or phrase. If the word or phrase is on the list provided by the annotators, we get 1 point. We divide this by the maximum Top-1 score, which is the number of test messages for which there was at least one word provided by the annotators. Notice that unlike entropy, Top-1 score cares about the order of results, but not about absolute values. In messages for which there is no correct answer, the Top-1 score is unaffected by the probabilities a system returns, while entropy measures give better scores to systems that return all low probabilities.

Finally, we compute the **Top-10** score. The Top-10 score is the number of words and phrases in the Top-10 which matched some word or phrase provided by the annotators, divided by the best possible Top-10 score (which is also the total number of annotations provided, since no message was given more than 10 words or phrases.)

	Top-1	Top-10
AA_{DF}-fixed query	10.86 ^{B}	30.56 ^{B}
AA_{DF}-fixed no query	4.87**	9.86**
AA_{DF} query	11.83^{B}	31.99 ^{B}
AA_{DF} no query	1.45**	9.65**
All-QL query	20.22^{B}	41.05^{B}
All –QL no query	10.07**	27.54**

Table 1: Query Restriction versus No Query

	Entropy	Top-1	Top-10
AA_{DF} fixed	0.3623 ^{B}	10.86 ^{B}	30.56 ^{B}
AA_{DF}	0.0279**	11.83	31.99*
$AA_{DF}S$ fixed	0.3670 ^{B}	12.684 ^{B}	32.141 ^{B}
$AA_{DF}S$	0.0267**	15.13	33.35
$AA_{DF}SS_{DF}$ fixed	0.3524^{B}	11.61 ^{B}	28.64 ^{B}
$AA_{DF}SS_{DF}$	0.0267**	15.32**	33.02**

Table 2: Fixed versus Automatic Parameters

3.2 Query Data Experiments

In this section, we present experiments showing that search engine query frequency data can dramatically improve our results. In particular, we used a restriction that all words returned were in the top 7.5 million most common queries to MSN Search. We tried several different model types, with and without the query frequency restriction. The different model types will be explained below.

In all of our results, we use a * to indicate statistical significance at the 95% level, and ** to indicate significance at the 99% level. We use a "^{B}" to indicate that a result is a baseline result for the results below it (until the next "^{B}"). Significance is tested with a two-tailed paired t-test. In Table 1, the test is done between the "query" result and the corresponding "no query" result. We do not show average entropy results in Table, because the number of test instances is different with and without the query: with the query restriction, there are many more test instances. This makes the entropies incomparable.

We now explain the models tested in Table 1. The first two lines compare "AA_{DF}-fixed query" to "AA_{DF}-fixed no query." "AA_{DF}" is a model that approximates a traditional TF-IDF model. We used the Anywhere and DFAnywhere features (AA_{DF}). Fixed means that the parameters were set to +1 and -1, making this essentially a TF-IDF model. "Query" means that the results were restricted to those in the top 7.5 million, while no query means that all results were allowed. As can be seen, the AA_{DF}-fixed query model significantly outperforms the no query model on both Top-1 and Top-10 measures. In the next two lines, we show results of training the parameters using logistic regression. AA_{DF} query is just like AA_{DF}-fixed query, except that the parameters are set with logistic regression, rather than being fixed at +1 and -1. Again, AA_{DF} query outperforms AA_{DF} no query. Next, we tried another model with all features (All), as will be described later, except for the query frequency features (QL), which we call All-QL. All-QL query (with the restriction) substantially outperforms All-QL no query (without the restriction.) Parameters were set with logistic regression.

In all cases but one, algorithms with the query restriction outperform algorithms without the restriction by at least a factor of 2 (and in that case, by a factor of 1.5) Because of both the increased speed, and the increased accuracy, all remaining experiments were done with the query file restriction.

3.3 Trained versus Fixed Parameters

Our next intuition was that setting parameters using training would be better than ad-hoc intuitive parameter settings. Logistic regression models are trained to optimize the entropy of training data, which is also equivalent to making the training data as likely as possible. Another way of saying this is that they try to do as good a job as possible of estimating the probabilities. The hope is that these results will also be good on Top-1 and Top-10 scores.

We ran three experiments comparing hand-set to automatically trained parameter settings. The first experiment used AA_{DF} as in the previous experiments. The second used $AA_{DF}S$ with settings of 1 for A, -1 for A_{DF} and 1 for S (subject line counts), similar to TF-IDF, but with a boost for words in the subject line. Finally, we tried $AA_{DF}SS_{DF}$ with a value of –1 for S_{DF}, similar to combining TF-IDF for all features, and TF-IDF for the subject.

The results are shown in Table 2. In all three cases, the entropy is much improved, which is what the training optimized. In addition, the Top-1 and Top-10 scores are also improved. (Astute readers might notice that the Top-1 and Top-10 scores for *AA_{DF}-fixed no query* are better than the scores for *AA_{DF} no query.* The "no query" case is extremely skewed, with a huge number of negative examples compared to a small number of positive examples. Because of this, training this well

	Entropy	Top-1	Top-10	Comments
all	0.0211^B	28.85^B	49.45^B	Baseline – all features
$noA_{DF}S_{DF}B_{DF}R_{DF}G_{DF}H_{DF}$	0.0241**	22.84**	39.66**	No Document Frequency
$noAA_{DF}$	0.0211	29.19	49.42	No "all"
$noSS_{DF}$	0.0215**	27.67	48.7	No Subject
$noBB_{DF}$	0.0211	28.55	49.25	No Body Only (not subject)
$noRR_{DF}$	0.0212	28.82	49.45	No Body-not-Reply
$noGG_{DF}$	0.0212	29.6	48.7	No At Beginning
$noHH_{DF}$	0.0212*	29.04	49.14	No Near Beginning
noQ	0.0234**	21.31**	44.04**	No Query Frequency
noL	0.0213**	28.86	48.96*	No Log-Query Frequency
noQL	0.0239**	20.22**	41.05**	No Query Frequency or Log-Query Frequency
noC	0.0221**	27.34	46.51*	No Capitalization
noT	0.0211	28.84	49.51	No Sentence Beginning
noP	0.0218**	27.73	47.23	No Phrase Length
noM	0.0217**	27.56	48.8	No Message Length
noF	0.0211	29.03	49.35	No Forward Subject
noE	0.0212	29.03	49.62	No Reply Subject
AA_{DF}	0.0279^B	11.83^B	31.99^B	Baseline – normal TF and DF features
$AA_{DF}SS_{DF}$	0.0267**	15.32*	33.02	Add Subject features
$AA_{DF}BB_{DF}$	0.0279	11.52	31.41	Add Body only features
$AA_{DF}RR_{DF}$	0.0277**	11.35	31.55	Add Body-not-Reply features
$AA_{DF}GG_{DF}$	0.0276**	11.62	31.61	Add At Beginning features
$AA_{DF}HH_{DF}$	0.0271**	12.79	32.04	Add Near Beginning features
$AA_{DF}Q$	0.0249**	24.82**	41.17**	Add Query features
$AA_{DF}L$	0.0268**	18.55**	37.83**	Add Log Query features
$AA_{DF}C$	0.0271**	13.29	36.14**	Add capitalization features
$AA_{DF}T$	0.0276**	12.47	31.39	Add Sentence Beginning features
$AA_{DF}P$	0.0276**	11.89	31.71	Add phrase length features
$AA_{DF}M$	0.0261**	13.26*	32.12	Add message length features
$AA_{DF}F$	0.0279*	11.83	31.99	Add has Forward Subject features
$AA_{DF}E$	0.0279	11.35	31.64	Add has Reply Subject features

Table 3: Contributions of Features

requires additional techniques, an area that will be described in future research.)

3.4 Individual Feature Contributions

Next, we performed a large number of experiments to show the contributions of individual features. We tried two sets of experiments. First, we started with a system combining all features, and then did ablation studies, removing various features, singly or in groups. Statistical significance of differences is measured relative to this baseline. Next, we started with a system with A and A_{DF} features (Standard TF-IDF features, with optimized weights), and then tried adding features.

Table 3 shows our results. In our ablation studies, no features, other than the query features, log query features, and in one case, the capitalization feature, had a statistically significant result on Top-1 or Top-10 score. Many of these features capture redundant information, and removing them one at a time has only a small impact. Also, Top-1 and Top-10 are somewhat noisy scores, and achieving siginificance is difficult. On the other hand, on the more sensitive entropy measure, we see that subject, query frequency, capitalization, phrase length, and message length all lead to statistically significant improvements. Of these, capitalization lead to the largest improvements.

Next, we started with a TF-IDF-like system (AA_{DF}) and tried adding features. Again, the largest improvements are from query frequency and capitalization. Message length has one of the largest impacts on entropy: for long messages, it can be difficult to pick out the few relevant keywords. Subject information is also noteworthy, leading to a moderate entropy reduction and a good-sized improvement on Top-1 score. Most of the other features lead to some small improvement

4 Related Work

There has been a moderate amount of related work in implicit query selection. The most used implicit query system in practice is probably Google's Gmail service, which in the past provided implicit queries for an email system: when reading Gmail messages, one used to be be presented with relevant advertisements or search results. (Today, only advertisements are shown.) However, to our knowledge, nothing has been made public about the workings of this system. Brin et al. (1998) state "A current very early prototype scans through email and retrieves relevant Web pages" although nothing else is said of the system. There is however, a more recent Google publication (Henzinger et al., 2003) on implicit queries for broadcast news services, using closed captioning information.

Henzinger et al. broke news stories into 15 second chunks, and tried to find print news articles relevant to the segment. They tried a number of techniques, including using IDF^2 instead of IDF; using stemming; and using a history feature. None of these features consistently improved performance. One fairly consistent improvement came from first trying three word queries, and then trying shorter queries if no results were returned. Our use of common queries may achieve something similar: users are unlikely to make queries that they expect will return no results.

Henzinger et al. also report research on postprocessing: taking the results of one or more searches and selecting the best results based on additional criteria that may include inspection of each returned document. They conclude that postprocessing is more important than query selection. In this paper, we do not examine postprocessing. For their application, finding queries for broadcast news, there would be a relatively small number of queries that need to be done, since at any given time, there are a relatively small number of broadcast news feeds. For our application, if widely deployed, there would be hundreds of millions, or perhaps even a billion queries a day performed; performing more queries than necessary and then doing time-consuming post processing is probably not practical. Still, some postprocessing and filtering, e.g. based on the summaries of the returned results, is an area for future research.

Dumais et al. (2004) examined implicit queries, in the context of an email system. Their work focused on user interface aspects of email implicit query. Interestingly, while email was the primary application of their work, they used a simple TF-IDF system that was not customized to email, e.g. not upweighting the importance of the subject.

Czerwinski et al. (1999) looked at implicit queries for web pages for retrieving related pages in a user's favorites; the primary emphasis was on the user interface aspects. They used two techniques to establish similarity: hand labelings of documents into clusters; and a simple cosine similarity between the word vectors after HTML and stopword removal.

Rhodes and Maes (2000) also examined the user interface aspects of an implicit query system. Their description of query matching is short and somewhat ambiguous, but it appears that they also use document similarity between query documents and target documents, weighted by IDF. This kind of matching can be done for small databases, but does not create a query that can be sent to a search engine like MSN Search or Google, nor is it easy to achieve for web-scale databases. Rhodes and Maes mention some of the problems email creates, such as signature lines, and possibly replies, but their solution is to remove such data before the query, in contrast to our solution of weighting different sections.

Budzik et al. (2001) examined an implicit query system for web pages and word processing documents. They primarily used word frequency to generate their queries. They used heuristics to augment the frequency metric, including removing stop words, valuing emphasized words, valuing words that occur at the beginning, and trying to ignore words in non-content sections (e.g. navigation bars on web pages.) No results are given showing whether these heuristics lead to improvements in practice. Note that Budzik et al. found a list of 20 words to submit to a search engine, relying on search engine behavior that would return the "best" match even if all words were not present. This functionality is not available in some modern search engines. Budzik used stop-words; which we did not, because we did not find stopwords to be a problem in our early experiments. However, some of the baseline systems we used might have benefited from stopwords.

5 Discussion

In this paper, we have made three main contributions.

First, we have shown the critical importance of including query frequency information, which appears to be a new idea. The query file restriction actually improved the Top-1 score by at least a factor of 2, and improved the Top-10 score typically by a factor of 1.5 or more. Also, it makes our code much more efficient: it is much easier to examine only the subset of words and phrases that are common queries than to examine all words. This is especially true if parameters are trained, since it substantially reduces the number of training examples.

Second, we have identified the most important set of features for an email implicit query system. This include the query frequency, message length, capitalization, and subject line information. All of these except for the subject line information could easily be applied to other document types. All of these except for the query frequency could be applied to other targets, such as relevant ads for a content-targeted advertising system. (For a content-targeted advertising system, we could presumably substitute the list of available ads and expected revenue or click-through rate for each ad.)

Third, we have shown how to optimize these parameters using logistic regression. This works at least as well as intuitive fixed values for the parameters, and allows us to combine the many features we have found to be helpful.

We see a large number of areas for future work. First, we believe that even more useful features can be found. Second, we are interested in exploring other learning algorithms. The learning for this case is somewhat different than typical classification problems for two reasons. First, the distribution is highly skewed, with thousands of negative examples for each positive one. Second, we are interested in Top-1 or Top-10 performance, rather than accuracy. Both of these problems open up opportunities for improved learning.

In addition, we focused here only on selecting the correct keywords. For an end-to-end application, retrieving the correct documents is also important. Henzinger et al. (2003) found that there were many potential improvements in this area, based on postprocessing. We are also interested in exploring this area in future work.

Overall, our results lead to large improvements. A reasonable baseline might be the "AA_{DF}-fixed no query" setting in Table 1, which achieves 4.9% and 9.9% Top-1 and Top-10 accuracy. The combination of our improvements leads to the "all" system of Table 3, which combines the query restriction, all features except phrase-length, and our machine learning system. It achieves 28.85% and 49.45% respectively on Top-1 and Top-10 measure – five or six times better than the baseline! In addition, our method returns the probabilities that queries are correct, which can be useful in real applications.

References

S. Brin, R. Motwani, L. Page, and T. Winograd (1998). What can you do with a web in your pocket? *Data Engineering Bulletin*, 21(2):37–47.

Budzik, J., Hammond, K. and Birnbaum, L. (2001). Information access in context. *Knowledge based systems, 14(1-2)*, 37-53.

Czerwinski, M., Dumais, S., Robertson, G., Dziadosz, S., Tiernan, S. and van Dantzich, M. (1999). "Visualizing implicit queries for information management and retrieval" In *Proceedings of CHI'99*, 560-567.

Chen, S. and Rosenfeld, R.. (1999) "A Gaussian prior for smoothing maximum entropy models." Technical Report CMUCS -99-108, Carnegie Mellon University.

Dumais, S., Cutrell, E., Sarin R., and Horvitz, E. (2004) "Implicit Queries (IQ) for Contextualized Search", Annual ACM Conference on Research and Development in Information Retrieval, Sheffield, UK.

Goodman, J. "Sequential Conditional Generalized Iterative Scaling" (2002), Association for Computational Linguistics, Philadelphia, Pennsylvania.

Google SEC Filing, August 2004.

Henzinger, M., Chang, B. W., Milch, B., and Brin, S. (2003) "Query-Free News Search". *Proc. 12th International World Wide Web Conference*.

Madden, M. and Rainie, L. (2003), "America's Online Pursuits". *Pew Internet and American Life Project.*

Rhodes, B. & Maes, P. (2000). "Just-in-time information retrieval agents" *IBM Systems Journal, 39(3-4)*, 685-704.

O. Dekel, C. Manning, Y. Singer. (2003) Log-Linear Models for Label Ranking. NIPS 2003.