Learning Dense Models of Query Similarity from User Click Logs

Fabio De Bona[*]
Friedrich Miescher Laboratory
of the Max Planck Society
Tübingen, Germany
fabio@tuebingen.mpg.de

Stefan Riezler
Google Research
Zürich, Switzerland
riezler@google.com

Keith Hall
Google Research
Zürich, Switzerland
kbhall@google.com

Massimiliano Ciaramita
Google Research
Zürich, Switzerland
massi@google.com

Amaç Herdağdelen[*]
University of Trento
Rovereto, Italy
amac@herdagdelen.com

Maria Holmqvist[*]
Linkopings University
Linkopings, Sweden
marho@ida.liu.se

Abstract

The goal of this work is to integrate query similarity metrics as features into a dense model that can be trained on large amounts of query log data, in order to rank query rewrites. We propose features that incorporate various notions of syntactic and semantic similarity in a generalized edit distance framework. We use the implicit feedback of user clicks on search results as weak labels in training linear ranking models on large data sets. We optimize different ranking objectives in a stochastic gradient descent framework. Our experiments show that a pairwise SVM ranker trained on multipartite rank levels outperforms other pairwise and listwise ranking methods under a variety of evaluation metrics.

1 Introduction

Measures of query similarity are used for a wide range of web search applications, including query expansion, query suggestions, or listings of related queries. Several recent approaches deploy user query logs to learn query similarities. One set of approaches focuses on user reformulations of queries that differ only in one phrase, e.g., Jones et al. (2006). Such phrases are then identified as candidate expansion terms, and filtered by various signals such as co-occurrence in similar sessions, or log-likelihood ratio of original and expansion phrase. Other approaches focus on the relation of queries and search results, either by clustering queries based on their search results, e.g., Beeferman and Berger (2000), or by deploying the graph of queries and results to find related queries, e.g., Sahami and Heilman (2006).

The approach closest to ours is that of Jones et al. (2006). Similar to their approach, we create a training set of candidate query rewrites from user query logs, and use it to train learners. While the dataset used in Jones et al. (2006) is in the order of a few thousand query-rewrite pairs, our dataset comprises around 1 billion query-rewrite pairs. Clearly, manual labeling of rewrite quality is not feasible for our dataset, and perhaps not even desirable. Instead, our intent is to learn from large amounts of user query log data. Such data permit to learn smooth models because of the effectiveness of large data sets to capture even rare aspects of language, and they also are available as *in the wild*, i.e., they reflect the actual input-output behaviour that we seek to automate (Halevy et al., 2009). We propose a technique to automatically create weak labels from co-click information in user query logs of search engines. The central idea is that two queries are related if they lead to user clicks on the same documents for a large amount of documents. A manual evaluation of a small subset showed that a determination of positive versus negative rewrites by thresholding the number of co-clicks correlates well with human judgements of similarity, thus justifying our method of eliciting labels from co-clicks.

Similar to Jones et al. (2006), the features of our models are not based on word identities, but instead on general string similarity metrics. This leads to dense rather than sparse feature spaces. The dif-

[*]The work presented in this paper was done while the authors were visiting Google Research, Zürich.

Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL, pages 474–482,
Los Angeles, California, June 2010. ©2010 Association for Computational Linguistics

ference of our approach to Jones et al. (2006) lies in our particular choice of string similarity metrics. While Jones et al. (2006) deploy "syntactic" features such as Levenshtein distance, and "semantic" features such as log-likelihood ratio or mutual information, we combine syntactic and semantic aspects into generalized edit-distance features where the cost of each edit operation is weighted by various term probability models.

Lastly, the learners used in our approach are applicable to very large datasets by an integration of linear ranking models into a stochastic gradient descent framework for optimization. We compare several linear ranking models, including a log-linear probability model for bipartite ranking, and pairwise and listwise SVM rankers. We show in an experimental evaluation that a pairwise SVM ranker trained on multipartite rank levels outperforms state-of-the-art pairwise and listwise ranking methods under a variety of evaluation metrics.

2 Query Similarity Measures

2.1 Semantic measures

In several of the similarity measures we describe below, we employ *pointwise mutual information* (PMI) as a measure of the association between two terms or queries. Let w_i and w_j be two strings that we want to measure the amount of association between. Let $\mathrm{p}(w_i)$ and $\mathrm{p}(w_j)$ be the probability of observing w_i and w_j in a given model; e.g., relative frequencies estimated from occurrence counts in a corpus. We also define $\mathrm{p}(w_i, w_j)$ as the joint probability of w_i and w_j; i.e., the probability of the two strings occurring together. We define PMI as follows:

$$\mathrm{PMI}(w_i, w_j) = \log \frac{\mathrm{p}(w_i, w_j)}{\mathrm{p}(w_i)\mathrm{p}(w_j)}. \qquad (1)$$

PMI has been introduced by Church and Hanks (1990) as word assosiatio ratio, and since then been used extensively to model semantic similarity. Among several desirable properties, it correlates well with human judgments (Recchia and Jones, 2009).

2.2 Taxonomic normalizations

As pointed out in earlier work, query transitions tend to correlate with taxonomic relations such as generalization and specialization (Lau and Horvitz, 1999; Rieh and Xie, 2006). Boldi et al. (2009) show how knowledge of transition types can positively impact query reformulation. We would like to exploit this information as well. However, rather than building a dedicated supervised classifier for this task we try to capture it directly at the source. First, we notice how string features; e.g., length, and edit distance already model this phenomenon to some extent, and in fact are part of the features used in Boldi et al. (2009). However, these measures are not always accurate and it is easy to find counterexamples both at the term level (e.g., "camping" to "outdoor activities" is a generalization) and character level ("animal pictures" to "cat pictures" is a specialization). Secondly, we propose that by manipulating PMI we can directly model taxonomic relations to some extent.

Rather than using raw PMI values we renormalize them. Notice that it is not obvious in our context how to interpret the relation between strings co-occurring less frequently than random. Such noisy events will yield negative PMI values since $p(w_i, w_j) < p(w_i)p(w_j)$. We enforce zero PMI values for such cases. If PMI is thus constrained to non-negative values, normalization will bound PMI to the range between 0 and 1.

The first type of normalization, called *joint normalization*, uses the negative log joint probability and is defined as

$$\mathrm{PMI(J)}(w_i, w_j) = \mathrm{PMI}(w_i, w_j)/{-\log}(\mathrm{p}(w_i, w_j)).$$

The jointly normalized PMI(J) is a symmetric measure between w_i and w_j in the sense that $\mathrm{PMI(J)}(w_i, w_j) = \mathrm{PMI(J)}(w_j, w_i)$. Intuitively it is a measure of the amount of shared information between the two strings relative to the sum of individual strings information. The advantages of the joint normalization of PMI have been noticed before (Bouma, 2009).

To capture asymmetries in the relation between two strings, we introduce two non-symmetric normalizations which also bound the measure between 0 and 1. The second normalization is called *specialization normalization* and is defined as

$$\mathrm{PMI(S)}(w_i, w_j) = \mathrm{PMI}(w_i, w_j)/ - \log(\mathrm{p}(w_i)).$$

The reason we call it specialization is that PMI(S) favors pairs where the second string is a specialization of the first one. For instance, PMI(S) is at its maximum when $\mathrm{p}(w_i, w_j) = \mathrm{p}(w_j)$ and that means the conditional probability $\mathrm{p}(w_i|w_j)$ is 1 which is an indication of a specialization relation.

The last normalization is called the *generalization normalization* and is defined in the reverse direction as

$$\mathrm{PMI(G)}(w_i, w_j) = \mathrm{PMI}(w_i, w_j)/ - \log(\mathrm{p}(w_j)).$$

Again, PMI(G) is a measure between 0 and 1 and is at its maximum value when $\mathrm{p}(w_j|w_i)$ is 1.

The three normalizations provide a richer representation of the association between two strings. Furthermore, jointly, they model in an information-theoretic sense the generalization-specialization dimension directly. As an example, for the query transition "apple" to "mac os" PMI(G)=0.2917 and PMI(S)=0.3686; i.e., there is more evidence for a specialization. Conversely for the query transition "ferrari models" to "ferrari" we get PMI(G)=1 and PMI(S)=0.5558; i.e., the target is a "perfect" generalization of the source[1].

2.3 Syntactic measures

Let V be a finite vocabulary and ξ be the null symbol. An edit operation: insertion, deletion or substitution, is a pair $(a, b) \in \{V \cup \{\xi\} \times V \cup \{\xi\}\} \setminus \{(\xi, \xi)\}$. An alignment between two sequences w_i and w_j is a sequence of edit operations $\omega = (a_1, b_1), ..., (a_n, b_n)$. Given a non-negative cost function c, the cost of an alignment is $c(\omega) = \sum_{i=1}^{n} c(\omega_i)$. The Levenshtein distance, or edit distance, defined over V, $d_V(w_i, w_j)$ between two sequences is the cost of the least expensive sequence of edit operations which transforms w_i into w_j (Levenshtein, 1966). The distance computation can be performed via dynamic programming in time $\mathrm{O}(|w_i||w_j|)$. Similarity at the string, i.e., character or term, level is an indicator of semantic similarity. Edit distance captures the amount of overlap between the queries as sequences of symbols and has been previously used in information retrieval (Boldi et al., 2009; Jones et al., 2006).

We use two basic Levenshtein distance models. The first, called Edit1 (E1), employs a unit cost function for each of the three operations. That is, given a finite vocabulary T containing all terms occurring in queries:

$$\forall a, b \in T, c_{\mathrm{E1}}(a, b) = 1 \text{ if}(a \neq b), 0 \text{ else}.$$

The second, called Edit2 (E2), uses unit costs for insertion and deletion, but computes the character-based edit distance between two terms to decide on the substitution cost. If two terms are very similar at the character level, then the cost of substitution is lower. Given a finite vocabulary T of terms and a finite vocabulary A of characters, the cost function is defined as:

$$\forall a, b \in T, c_{E2}(a, b) = d_A(a, b) \text{ if} a \wedge b \neq \xi, 1 \text{ else}.$$

where $d_A(a, b)$ is linearly scaled between 0 and 1 dividing by $\max(|a|, |b|)$.

We also investigate a variant of the edit distance algorithm in which the terms in the input sequences are sorted, alphabetically, before the distance computation. The motivation behind this variant is the observation that linear order in queries is not always meaningful. For example, it seems reasonable to assume that "brooklyn pizza" and "pizza brooklyn" denote roughly the same user intent. However, the pair has an edit distance of two (delete-insert), while the distance between "brooklyn pizza" and the less relevant "brooklyn college" is only one (substitute). The sorted variant relaxes the ordering constraint.

2.4 Generalized measures

In this section we extend the edit distance framework introduced in Section 2.3 with the semantic similarity measures described in Section 2.1, using the taxonomic normalizations defined in Section 2.2.

Extending the Levenshtein distance framework to take into account semantic similarities between terms is conceptually simple. As in the Edit2 model above we use a modified cost function. We introduce a *cost matrix* encoding individual costs for term substitution operations; the cost is defined in terms of the normalized PMI measures of Section 2.2, recall that these measures range between 0 and 1. Given a normalized similarity measure f, an entry in a cost matrix S for a term pair (w_i, w_j) is defined as:

$$s(w_i, w_j) = 2 - 2f(w_i, w_j) + \epsilon$$

[1]The values are computed from Web counts.

We call these models SEdit (SE), where S specifies the cost matrix used. Given a finite term vocabulary T and cost matrix S, the cost function is defined as:

$$\forall a, b \in T, c_{SE}(a, b) = s(a, b) \text{ if} a \wedge b \neq \xi, 1 \text{ else.}$$

The cost function has the following properties. Since insertion and deletion have unit cost, a term is substituted only if a substitution is "cheaper" than deleting and inserting another term, namely, if the similarity between the terms is not zero. The ϵ correction, coupled with unit insertion and deletion cost, guarantees that for an unrelated term pair a combination of insertion and deletion will always be less costly then a substitution. Thus in the computation of the optimal alignment, each operation cost ranges between 0 and 2.

As a remark on efficiency, we notice that here the semantic similarities are computed between terms, rather than full queries. At the term level, caching techniques can be applied more effectively to speed up feature computation. The cost function is implemented as a pre-calculated matrix, in the next section we describe how the matrix is estimated.

2.5 Cost matrix estimation

In our experiments we evaluated two different sources to obtain the PMI-based cost matrices. In both cases, we assumed that the cost of the substitution of a term with itself (i.e. identity substitution) is always 0. The first technique uses a probabilistic clustering model trained on queries and clicked documents from user query logs. The second model estimates cost matrices directly from user session logs, consisting of approximately 1.3 billion U.S. English queries. A session is defined as a sequence of queries from the same user within a controlled time interval. Let q_s and q_t be a query pair observed in the session data where q_t is issued immediately after q_s in the same session. Let $q'_s = q_s \setminus q_t$ and $q'_t = q_t \setminus q_s$, where $\setminus$ is the set difference operator. The co-occurrence count of two terms w_i and w_j from a query pair q_s, q_t is denoted by $n_{i,j}(q_s, q_t)$ and is defined as:

$$n_{i,j}(q_s, q_t) = \begin{cases} 1 \text{ if } w_i = w_j \wedge w_i \in q_s \wedge w_j \in q_t \\ 1/(|q'_s|\,|q'_t|) \text{ if } w_i \in q'_s \wedge w_j \in q'_t \\ 0 \text{ else.} \end{cases}$$

In other words, if a term occurs in both queries, it has a co-occurrence count of 1. For all other term pairs, a normalized co-occurrence count is computed in order to make sure the sum of co-occurrence counts for a term $w_i \in q_s$ sums to 1 for a given query pair. The normalization is an attempt to avoid the under representation of terms occurring in both queries.

The final co-occurrence count of two arbitrary terms w_i and w_j is denoted by $N_{i,j}$ and it is defined as the sum over all query pairs in the session logs, $N_{i,j} = \sum_{q_s,q_t} n_{i,j}(q_s, q_t)$. Let $N = \sum_{w_i,w_j} N_{i,j}$ be the sum of co-occurrence counts over all term pairs. Then we define a joint probability for a term pair as $\mathrm{p}(w_i, w_j) = \frac{N_{i,j}}{N}$. Similarly, we define the single-occurrence counts and probabilities of the terms by computing the marginalized sums over all term pairs. Namely, the probability of a term w_i occurring in the source query is $\mathrm{p}(i, \cdot) = \sum_{w_j} N_{i,j}/N$ and similarly the probability of a term w_j occurring in the target query is $\mathrm{p}(\cdot, j) = \sum_{w_i} N_{i,j}/N$. Plugging in these values in Eq. (1), we get the $\mathrm{PMI}(w_i, w_j)$ for term pair w_i and w_j, which are further normalized as described in Section 2.2.

More explanation and evaluation of the features described in this section can be found in Ciaramita et al. (2010).

3 Learning to Rank from Co-Click Data

3.1 Extracting Weak Labels from Co-Clicks

Several studies have shown that implicit feedback from clickstream data is a weaker signal than human relevance judgements. Joachims (2002) or Agrawal et al. (2009) presented techniques to convert clicks into labels that can be used for machine learning. Our goal is not to elicit relevance judgments from user clicks, but rather to relate queries by pivoting on commonly clicked search results. The hypothesis is that two queries are related if they lead to user clicks on the same documents for a large amount of documents. This approach is similar to the method proposed by Fitzpatrick and Dent (1997) who attempt to measure the relatedness between two queries by using the normalized intersection of the top 200 retrieval results. We add click information to this setup, thus strengthening the preference for precision over recall in the extraction of related queries.

Table 1: Statistics of co-click data sets.

	train	dev	test
number of queries	250,000	2,500	100
average number of rewrites per query	4,500	4,500	30
percentage of rewrites with $\geq$ 10 coclicks	0.2	0.2	43

In our experiments we created two ground-truth ranking scenarios from the co-click signals. In a first scenario, called *bipartite ranking*, we extract a set of positive and a set of negative query-rewrite pairs from the user logs data. We define positive pairs as queries that have been co-clicked with at least 10 different results, and negative pairs as query pairs with fewer than 10 co-clicks. In a second scenario, called *multipartite ranking*, we define a hierarchy of levels of "goodness", by combining rewrites with the same number of co-clicks at the same level, with increasing ranks for higher number of co-clicks. Statistics on the co-click data prepared for our experiments are given in Table 1.

For training and development, we collected query-rewrite pairs from user query logs that contained at least one positive rewrite. The training set consists of about 1 billion of query-rewrite pairs; the development set contains 10 million query-rewrite pairs. The average number of rewrites per query is around 4,500 for the training and development set, with a very small amount of 0.2% positive rewrites per query. In order to confirm the validity of our co-click hypothesis, and for final evaluation, we held out another sample of query-rewrite pairs for manual evaluation. This dataset contains 100 queries for each of which we sampled 30 rewrites in descending order of co-clicks, resulting in a high percentage of 43% positive rewrites per query. The query-rewrite pairs were annotated by 3 raters as follows: First the raters were asked to rank the rewrites in descending order of relevance using a graphical user interface. Second the raters assigned rank labels and binary relevance scores to the ranked list of rewrites. This labeling strategy is similar to the labeling strategy for synonymy judgements proposed by Rubenstein and Goodenough (1965). Inter-rater agreements on binary relevance judgements, and agreement between rounded averaged human relevance scores and assignments of positive/negative labels by the co-click threshold of 10 produced a Kappa value of 0.65 (Siegel and Castellan, 1988).

3.2 Learning-to-Rank Query Rewrites

3.2.1 Notation

Let $S = \{(x_q, y_q)\}_{q=1}^n$ be a training sample of queries, each represented by a set of rewrites $x_q = \{x_{q1}, \ldots, x_{q,n(q)}\}$, and set of rank labels $y_q = \{y_{q1}, \ldots, y_{q,n(q)}\}$, where $n(q)$ is the number of rewrites for query q. For full rankings of all rewrites for a query, a total order on rewrites is assumed, with rank labels taking on values $y_{qi} \in \{1, \ldots, n(q)\}$. Rewrites of equivalent rank can be specified by assuming a partial order on rewrites, where a multipartite ranking involves $r < n(q)$ relevance levels such that $y_{qi} \in \{1, \ldots, r\}$, and a bipartite ranking involves two rank values $y_{qi} \in \{1, 2\}$ with relevant rewrites at rank 1 and non-relevant rewrites at rank 2.

Let the rewrites in x_q be identified by the integers $\{1, 2, \ldots, n(q)\}$, and let a permutation π_q on x_q be defined as a bijection from $\{1, 2, \ldots, n(q)\}$ onto itself. Let Π_q denote the set of all possible permutations on x_q, and let π_{qi} denote the rank position of x_{qi}. Furthermore, let (i, j) denote a pair of rewrites in x_q and let $\mathcal{P}_q$ be the set of all pairs in x_q.

We associate a feature function $\phi(x_{qi})$ with each rewrite $i = 1, \ldots, n(q)$ for each query q. Furthermore, a partial-order feature map as used in Yue et al. (2007) is created for each rewrite set as follows:

$$\phi(x_q, \pi_q) = \frac{1}{|\mathcal{P}_q|} \sum_{(i,j) \in \mathcal{P}_q} \phi(x_{qi}) - \phi(x_{qj}) \mathrm{sgn}(\frac{1}{\pi_{qi}} - \frac{1}{\pi_{qj}}).$$

The goal of learning a ranking over the rewrites x_q for a query q can be achieved either by sorting the rewrites according to the rewrite-level ranking function $f(x_{qi}) = \langle w, \phi(x_{qi}) \rangle$, or by finding the permutation that scores highest according to a query-level ranking function $f(x_q, \pi_q) = \langle w, \phi(x_q, \pi_q) \rangle$.

In the following, we will describe a variety of well-known ranking objectives, and extensions thereof, that are used in our experiments. Optimization is done in a stochastic gradient descent (SGD) framework. We minimize an empirical loss objective

$$\min_w \sum_{x_q, y_q} \ell(w)$$

by stochastic updating

$$w_{t+1} = w_t - \eta_t g_t$$

where η_t is a learning rate, and g_t is the gradient

$$g_t = \nabla \ell(w)$$

where

$$\nabla \ell(w) = \left\langle \frac{\partial}{\partial w_1}\ell(w), \frac{\partial}{\partial w_2}\ell(w), \ldots, \frac{\partial}{\partial w_n}\ell(w) \right\rangle.$$

3.2.2 Listwise Hinge Loss

Standard ranking evaluation metrics such as (Mean) Average Precision (Manning et al., 2008) are defined on permutations of whole lists and are not decomposable over instances. Joachims (2005), Yue et al. (2007), or Chakrabarti et al. (2008) have proposed multivariate SVM models to optimize such listwise evaluation metrics. The central idea is to formalize the evaluation metric as a prediction loss function L, and incorporate L via margin rescaling into the hinge loss function, such that an upper bound on the prediction loss is achieved (see Tsochantaridis et al. (2004), Proposition 2).

The loss function is given by the following listwise hinge loss:

$$\ell_{lh}(w) = (L(y_q, \pi_q^*) - \langle w, \phi(x_q, y_q) - \phi(x_q, \pi_q^*)\rangle)_+$$

where π_q^* is the maximizer of the $\max_{\pi_q \in \Pi_q \setminus y_q} L(y_q, \pi_q^*) + \langle w, \phi(x_q, \pi_q^*)\rangle$ expression, $(z)_+ = \max\{0, z\}$ and $L(y_q, \pi_q) \in [0, 1]$ denotes a prediction loss of a predicted ranking π_q compared to the ground-truth ranking y_q.[2]

In this paper, we use Average Precision (AP) as prediction loss function s.t.

$$L_{AP}(y_q, \pi_q) = 1 - AP(y_q, \pi_q)$$

where AP is defined as follows:

$$\begin{aligned} AP(y_q, \pi_q) &= \frac{\sum_{j=1}^{n(q)} Prec(j) \cdot (|y_{qj} - 2|)}{\sum_{j=1}^{n(q)} (|y_{qj} - 2|)}, \\ Prec(j) &= \frac{\sum_{k:\pi_{qk} \leq \pi_{qj}} (|y_{qk} - 2|)}{\pi_{qj}}. \end{aligned}$$

[2]We slightly abuse the notation y_q to denote the permutation on x_q that is induced by the rank labels. In case of full rankings, the permutation π_q corresponding to ranking y_q is unique. For multipartite and bipartite rankings, there is more than one possible permutation for a given ranking, so that we let π_q denote a permutation that is consistent with ranking y_q.

Note that the ranking scenario is in this case bipartite with $y_{qi} \in \{1, 2\}$.

The derivatives for ℓ_{lh} are as follows:

$$\frac{\partial}{\partial w_k}\ell_{lh} = \begin{cases} 0 \text{ if } \; (\langle w, \phi(x_q, y_q) - \phi(x_q, \pi_q^*)\rangle) \\ \qquad\qquad > L(y_q, \pi_q^*), \\ -(\phi_k(x_q, y_q) - \phi_k(x_q, \pi_q^*)) \text{ else.} \end{cases}$$

SGD optimization involves computing π_q^* for each feature and each query, which can be done efficiently using the greedy algorithm proposed by Yue et al. (2007). We will refer to this method as the *SVM-MAP* model.

3.2.3 Pairwise Hinge Loss for Bipartite and Multipartite Ranking

Joachims (2002) proposed an SVM method that defines the ranking problem as a pairwise classification problem. Cortes et al. (2007) extended this method to a magnitude-preserving version by penalizing a pairwise misranking by the magnitude of the difference in preference labels. A position-sensitive penalty for pairwise ranking SVMs was proposed by Riezler and De Bona (2009) and Chapelle and Keerthi (2010), and earlier for perceptrons by Shen and Joshi (2005). In the latter approaches, the magnitude of the difference in inverted ranks is accrued for each misranked pair. The idea is to impose an increased penalty for misrankings at the top of the list, and for misrankings that involve a difference of several rank levels.

Similar to the listwise case, we can view the penalty as a prediction loss function, and incorporate it into the hinge loss function by rescaling the margin by a pairwise prediction loss function $L(y_{qi}, y_{qj})$. In our experiments we used a position-sensitive prediction loss function

$$L(y_{qi}, y_{qj}) = \left|\frac{1}{y_{qi}} - \frac{1}{y_{qj}}\right|$$

defined on the difference of inverted ranks. The margin-rescaled pairwise hinge loss is then defined as follows:

$$\begin{aligned} \ell_{ph}(w) \;=\; & \sum_{(i,j) \in \mathcal{P}_q} (L(y_{qi}, y_{qj}) - \\ & \langle w, \phi(x_{qi}) - \phi(x_{qj})\rangle \operatorname{sgn}(\frac{1}{y_{qi}} - \frac{1}{y_{qj}}))_+ \end{aligned}$$

Table 2: Experimental evaluation of *random* and *best feature* baselines, and *log-linear*, *SVM-MAP*, *SVM-bipartite*, *SVM-multipartite*, and *SVM-multipartite-margin-rescaled* learning-to-rank models on manually labeled test set.

	MAP	NDCG@10	AUC	Prec@1	Prec@3	Prec@5
Random	51.8	48.7	50.4	45.6	45.6	46.6
Best-feature	71.9	70.2	74.5	70.2	68.1	68.7
SVM-bipart.	73.7	73.7	74.7	79.4	70.1	70.1
SVM-MAP	74.3	75.2	75.3	76.3	71.8	72.0
Log-linear	74.7	75.1	75.7	75.3	72.2	71.3
SVM-pos.-sens.	75.7	76.0	76.6	82.5	72.9	73.0
SVM-multipart.	**76.5**	**77.3**	**77.2**	**83.5**	**74.2**	**73.6**

The derivative of ℓ_{ph} is calculated as follows:

$$\frac{\partial}{\partial w_k}\ell_{lp} = \begin{cases} 0 \text{ if } (\langle w, \phi(x_{qi}) - \phi(x_{qj})\rangle \\ \qquad \text{sgn}(\frac{1}{y_{qi}} - \frac{1}{y_{qj}})) > L(y_{qi}, y_{qj}), \\ -(\phi_k(x_{qi}) - \phi_k(x_{qj}))\text{sgn}(\frac{1}{y_{qi}} - \frac{1}{y_{qj}}) \\ \quad \text{else.} \end{cases}$$

Note that the effect of inducing a position-sensitive penalty on pairwise misrankings applies only in case of full rankings on $n(q)$ rank levels, or in case of multipartite rankings involving $2 < r < n(q)$ rank levels. Henceforth we will refer to margin-rescaled pairwise hinge loss for multipartite rankings as the *SVM-pos.-sens.* method.

Bipartite ranking is a special case where $L(y_{qi}, y_{qj})$ is constant so that margin rescaling does not have the effect of inducing position-sensitivity. This method will be referred to as the *SVM-bipartite* model.

Also note that for full ranking or multipartite ranking, predicting a low ranking for an instance that is ranked high in the ground truth has a domino effect of accruing an additional penalty at each rank level. This effect is independent of margin-rescaling. The method of pairwise hinge loss for multipartite ranking with constant margin will henceforth be referred to as the *SVM-multipartite* model.

Computation in SGD optimization is dominated by the number of pairwise comparisons $|\mathcal{P}_q|$ for each query. For full ranking, a comparison of $|\mathcal{P}_q| = \binom{n(q)}{2}$ pairs has to be done. In the case of multipartite ranking at r rank levels, each including $|l_i|$ rewrites, pairwise comparisons between rewrites at the same rank level can be ignored. This reduces the number of comparisons to $|\mathcal{P}_q| = \sum_{i=1}^{r-1}\sum_{j=i+1}^{r} |l_i||l_j|$. For bipartite ranking of p positive and n negative instances, $|\mathcal{P}_q| = p \cdot n$ comparisons are necessary.

3.2.4 Log-linear Models for Bipartite Ranking

A probabilistic model for bipartite ranking can be defined as the conditional probability of the set of relevant rewrites, i.e., rewrites at rank level 1, given all rewrites at rank levels 1 and 2. A formalization in the family of log-linear models yields the following logistic loss function ℓ_{llm} that was used for discriminative estimation from sets of partially labeled data in Riezler et al. (2002):

$$\ell_{llm}(w) = -\log \frac{\sum_{x_{qi}\in x_q | y_{qi}=1} e^{\langle w, \phi(x_{qi})\rangle}}{\sum_{x_{qi}\in x_q} e^{\langle w, \phi(x_{qi})\rangle}}.$$

The gradient of ℓ_{llm} is calculated as a difference between two expectations:

$$\frac{\partial}{\partial w_k}\ell_{llm} = -p_w\left[\phi_k | x_q; y_{qi} = 1\right] + p_w\left[\phi_k | x_q\right].$$

The SGD computation for the log-linear model is dominated by the computation of expectations for each query. The logistic loss for bipartite ranking is henceforth referred to as the *log-linear* model.

4 Experimental Results

In the experiments reported in this paper, we trained linear ranking models on 1 billion query-rewrite pairs using 60 dense features, combined of the building blocks of syntactic and semantic similarity metrics under different estimations of cost matrices. Development testing was done on a data set that was held-out from the training set. Final testing was carried out on the manually labeled dataset. Data statistics for all sets are given in Table 1.

Table 3: P-values computed by approximate randomization test for 15 pairwise comparisons of result differences.

	Best-feature	SVM-bipart.	SVM-MAP	Log-linear	SVM-pos.-sens.	SVM-multipart.
Best-feature	-	< 0.005	< 0.005	< 0.005	< 0.005	< 0.005
SVM-bipart.	-	-	$\mathbf{0.324}$	< 0.005	< 0.005	< 0.005
SVM-MAP	-	-	-	$\mathbf{0.374}$	< 0.005	< 0.005
Log-linear	-	-	-	-	$\mathbf{0.053}$	< 0.005
SVM-pos.-sens.	-	-	-	-	-	< 0.005
SVM-multipart.	-	-	-	-	-	-

Model selection was performed by adjusting meta-parameters on the development set. We trained each model at constant learning rates $\eta \in \{1, 0.5, 0.1, 0.01, 0.001\}$, and evaluated each variant after every fifth out of 100 passes over the training set. The variant with the highest MAP score on the development set was chosen and evaluated on the test set. This early stopping routine also served for regularization.

Evaluation results for the systems are reported in Table 2. We evaluate all models according to the following evaluation metrics: Mean Average Precision (MAP), Normalized Discounted Cumulative Gain with a cutoff at rank 10 (NDCG@10), Area-under-the-ROC-curve (AUC), Precision@n[3]. As baselines we report a random permutation of rewrites (*random*), and the single dense feature that performed best on the development set (*best-feature*). The latter is the log-probability assigned to the query-rewrite pair by the probabilistic clustering model used for cost matrix estimation (see Section 2.5). P-values are reported in Table 3 for all pairwise comparisons of systems (except the random baseline) using an Approximate Randomization test where stratified shuffling is applied to results on the query level (see Noreen (1989)). The rows in Tables 2 and 3 are ranked according to MAP values of the systems. *SVM-multipartite* outperforms all other ranking systems under all evaluation metrics at a significance level ≥ 0.995. For all other pairwise comparisons of result differences, we find result differences of systems ranked next to each other to be not statistically significant. All systems outperform the *random* and *best-feature* baselines with statistically significant result differences. The distinctive advantage of the *SVM-multipartite* models lies in the possibility to rank rewrites with very high co-click numbers even higher than rewrites with reasonable numbers of co-clicks. This preference for ranking the top co-clicked rewrites high seems the best avenue for transferring co-click information to the human judgements encoded in the manually labeled test set. Position-sensitive margin rescaling does not seem to help, but rather seems to hurt.

[3]For a definition of these metrics see Manning et al. (2008)

5 Discussion

We presented an approach to learn rankings of query rewrites from large amounts of user query log data. We showed how to use the implicit co-click feedback about rewrite quality in user log data to train ranking models that perform well on ranking query rewrites according to human quality standards. We presented large-scale experiments using SGD optimization for linear ranking models. Our experimental results show that an SVM model for multipartite ranking outperforms other linear ranking models under several evaluation metrics. In future work, we would like to extend our approach to other models, e.g., sparse combinations of lexicalized features.

References

R. Agrawal, A. Halverson, K. Kenthapadi, N. Mishra, and P. Tsaparas. 2009. Generating labels from clicks. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining*, Barcelona, Spain.

Doug Beeferman and Adam Berger. 2000. Agglomerative clustering of a search engine query log. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*, Boston, MA.

P. Boldi, F. Bonchi, C. Castillo, and S. Vigna. 2009. From 'Dango' to 'Japanese cakes': Query reformulation models and patterns. In *Proceedings of Web Intelligence*. IEEE Cs Press.

G. Bouma. 2009. Normalized (pointwise) mutual information in collocation extraction. In *Proceedings of GSCL*.

Soumen Chakrabarti, Rajiv Khanna, Uma Sawant, and Chiru Bhattacharayya. 2008. Structured learning for non-smooth ranking losses. In *Proceedings of the 14th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'08)*, Las Vegas, NV.

Olivier Chapelle and S. Sathiya Keerthi. 2010. Efficient algorithms for ranking with SVMs. *Information Retrieval Journal*.

Kenneth Church and Patrick Hanks. 1990. Word association norms, mutual information and lexicography. *Computational Linguistics*, 16(1):22–29.

Massimiliano Ciaramita, Amaç Herdağdelen, Daniel Mahler, Maria Holmqvist, Keith Hall, Stefan Riezler, and Enrique Alfonseca. 2010. Generalized syntactic and semantic models of query reformulation. In *Proceedings of the 33rd ACM SIGIR Conference*, Geneva, Switzerland.

Corinna Cortes, Mehryar Mohri, and Asish Rastogi. 2007. Magnitude-preserving ranking algorithms. In *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, Corvallis, OR.

Larry Fitzpatrick and Mei Dent. 1997. Automatic feedback using past queries: Social searching? In *Proceedings of the 20th Annual International ACM SIGIR Conference*, Philadelphia, PA.

Alon Halevy, Peter Norvig, and Fernando Pereira. 2009. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24:8–12.

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'08)*, New York, NY.

Thorsten Joachims. 2005. A support vector method for multivariate performance measures. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*, Bonn, Germany.

Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. 2006. Generating query substitutions. In *Proceedings of the 15th International World Wide Web conference (WWW'06)*, Edinburgh, Scotland.

T. Lau and E. Horvitz. 1999. Patterns of search: analyzing and modeling web query refinement. In *Proceedings of the seventh international conference on User modeling*, pages 119–128. Springer-Verlag New York, Inc.

V.I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

Eric W. Noreen. 1989. *Computer Intensive Methods for Testing Hypotheses. An Introduction*. Wiley, New York.

G. Recchia and M.N. Jones. 2009. More data trumps smarter algorithms: comparing pointwise mutual information with latent semantic analysis. *Behavioral Research Methods*, 41(3):647–656.

S.Y. Rieh and H. Xie. 2006. Analysis of multiple query reformulations on the web: the interactive information retrieval context. *Inf. Process. Manage.*, 42(3):751–768.

Stefan Riezler and Fabio De Bona. 2009. Simple risk bounds for position-sensitive max-margin ranking algorithms. In *Proceedings of the Workshop on Advances in Ranking at the 23rd Annual Conference on Neural Information Processing Systems (NIPS'09)*, Whistler, Canada.

Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia, PA.

Herbert Rubenstein and John B. Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 10(3):627–633.

Mehran Sahami and Timothy D. Heilman. 2006. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th International World Wide Web conference (WWW'06)*, Edinburgh, Scotland.

Libin Shen and Aravind K. Joshi. 2005. Ranking and reranking with perceptron. *Journal of Machine Learning Research*, 60(1-3):73–96.

Sidney Siegel and John Castellan. 1988. *Nonparametric Statistics for the Behavioral Sciences. Second Edition*. MacGraw-Hill, Boston, MA.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*, Banff, Canada.

Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. 2007. A support vector method for optimizing average precision. In *Proceedings of the 30th Annual International ACM SIGIR Conference*, Amsterdam, The Netherlands.