Efficient and Accurate Local Learning for Ranking

Somnath Banerjee
HP Labs India

Avinava Dubey
IIT Bombay

Jinesh Machchhar
IIT Bombay

Soumen Chakrabarti
IIT Bombay

ABSTRACT

Learning to score—and thereby order—items represented as feature vectors $x_i \in \mathbb{R}^d$, with the goal of minimizing various ranking loss functions, is now a major topic in Information Retrieval. Training data consists of relevant and irrelevant documents identified for a number of queries. Many systems train a model $w \in \mathbb{R}^d$, such that the score of an item x_i is $w^\top x_i$. However, queries are diverse: navigational, informational, transactional, etc. Recently, there has been interest in local learning: customizing a model to each test query. While intuitively appealing, these proposals have either resulted in modest gains, or entail excessive computational burden at test time. In this paper we propose a local learning algorithm based on a new similarity measure between queries. The proposed local learning algorithm does not depend on a fixed query classification scheme. First, we represent (relevant and irrelevant) document vectors for the query as a point cloud. Second, we define an intuitive notion of similarity between the shapes of two point clouds, based on principal component analysis (PCA). Our local learning algorithm clusters queries at training time, using the PCA-based measure of query similarity. During test time, we simply locate the nearest training cluster, and use the model trained for that cluster. Very few clusters are adequate to give substantial boost to test accuracy. Our test time is small, training time is reasonable, and our accuracy beats several recent local learning approaches, as tested on the well-known LETOR dataset.

Categories and Subject Descriptors: H.3.3 [**Information Search and Retrieval**]: Retrieval models
General Terms: Algorithms, Experimentation
Keywords: Learning to rank, Local Ranking

1. INTRODUCTION

Learning to rank is an area of Machine Learning that has seen much recent activity. In the most common scenario, the training data consists of a set of queries Q. Each query $q \in Q$ is associated with a set of document D_q, represented as feature vectors $x_{qi} \in \mathbb{R}^d$. (We describe the elements of these vectors shortly.) Each document x_{qi} is associated with a relevance label z_{qi}. Often, the relevance label is binary, i.e., the document is relevant ("good") or irrelevant ("bad"), but three or more levels of relevance are also possible. A large class of algorithms for learning to rank uses this training data to estimate a *scoring model* $w \in \mathbb{R}^d$. When deployed, a test query t is submitted to the system, along with its document set D_t. The system then assigns a *score* $w^\top x_{ti}$ to each feature vector $x_{ti} \in D_t$, and then the documents are sorted in decreasing order of score.

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.
Copyright 2009 ACM X-XXXXX-XX-X/XX/XX ...$10.00.

Most systems that learn to rank use a few standard families of features in the document feature vectors x_{qi}. Each element in the vector x_{qi} represents some notion of match between the query q and the document indexed by i, or some notion of quality of document i.

As pointed out by several researchers [2, 8, 5], Web queries are of various types. Two major types are *navigational*, where a definite authoritative URL like `http://www.ibm.com` is sought, and *informational*, where the user wants to satisfy an information need by browsing many relevant pages. Different families of features have different impacts on ranking for different query classes. For navigational queries, the URL substring and PageRank features may be the best indicators of quality, whereas for informational queries, whole-page TFIDF and BM25 features may be better. In other words, a single model w may not be adequate as a scoring (hence ranking) device for diverse types of queries. That is where local learning comes in.

1.1 Related Work

In recent years, researchers are increasingly proposing that a single model w may not be sufficiently expressive to reduce ranking loss to the maximum extent possible (but this may not be entirely attributable to query diversity alone). Here we review prior work with this general flavor, and discuss two closely relate papers [8, 5] in detail.

1.1.1 Composite ranking models

Algorithms for learning ranking models may be trained using itemwise, pairwise or listwise objectives. Itemwise learners seek to regress x_{qi} to relevance label z_{qi}, perhaps via ordinal regression [4]. A recent regression algorithm uses gradient boosted regression trees [10], which can express highly nonlinear scoring functions. However, this work does not take query diversity into account.

RankBoost [7] is a pairwise trainer that estimates an overall scoring function $H(x) = \sum_m \alpha_m h_m(x)$ as a linear combination of weak scoring functions. This composition can express nonlinearity only if some weak functions $h_m(x)$ are nonlinear in feature vector x.

If the application requires $K > 2$ levels of relevance, Qin *et al.* [12] propose to learn $\binom{K}{2}$ ranking model, one for each pair of relevance levels. Then they use a weighted Borda count to aggregate ranking information across all the models. While this is a composite ranking system, query diversity is not the focus here.

1.1.2 PCA and local projection of training data

Duh and Kirchhoff [5] present a different lazy approach to local learning for ranking. They do nothing at training time. At test time, given a test query t they compute a kernel principal component [1] on the feature vectors of D_t. Principal Component Analysis (PCA) finds the directions of maximum variation for a given set of points, by computing the eigenvectors of the covariance matrix. They retain P principal components $\{u_1, \ldots, u_P\}$ corresponding to the top P eigenvalues. D&K's intuition was that these direc-

tions capture the most important directions in the feature space of documents associated with test query t. Therefore, they *project* all training data on to the subspace U defined by $u_1, \ldots, u_P$ and create a locally-tuned representation of training data. Then a model is trained on this projected training data to rank the documents of test query t.

The main problem with the D&K proposal is that the entire training work is deferred to test time. For the small LETOR data sets [11], test time was tolerable. But test time scales with the total number of training documents, which is clearly unacceptable.

1.1.3 Nearest-neighbor query clusters

Geng *et al.* [8] represent each *query* as a 27-dimensional feature vector. These features are derived from the documents associated with the query. For a document they define 27 query-derived features as described in [13]. A query is represented by taking average of those features from the top T documents ordered by BM25 score.

For each query in the training set a seperate ranking model is trained using the query and its K nearest queries. Here the distance between queries is the Euclidean distance in the 27-dimensional embedding of queries. Given a test query, they find the nearest training query and use the model associated with that query.

Their test time is reasonably fast if the training set is of manageable size, although it does scale with training set size. Training time is quite large, because the number of models to be trained is equal to the number of training queries. In our experiments, accuracy gains beyond a competitive single-model baseline were quite modest. They did not use LETOR because they found the number of queries in LETOR too small to see any benefit from local learning (interestingly, this turns out not to be the case for our approach). They used proprietary data sets, and obtained some improvement in NDCG. They did not report on the scalability and run time performance of their training and testing algorithms.

1.2 Our contributions

We make three contributions in this paper:

- We propose a new representation of queries and a notion of similarity between queries using this representation.
- We propose an offline method to cluster queries based on this similarity, and train a model for each cluster. Our training time is reasonable compared to a single model training time and orders of magnitude smaller than Geng *et al.*. Given a test query, we use the model from the most similar cluster, also defined suitably. This gives a considerable accuracy boost compared to baselines, Geng *et al.*'s approach, and Duh *et al.*'s approach.
- To improve test time scaling as training size improves, we adapt locality sensitive hashtables (LSH) in local learning to rank. We show that this results in significant reduction of testing time. Our test time is orders of magnitude smaller than Duh *et al.*, while achieving better test accuracy.

In the next section we describe different methods of representing a query as a point cloud. We propose our local learning algorithm in Section 3 and in Section 4 we report on extensive experiments with the LETOR (version 3) data sets [11] as well as a new data set released by a Russian Internet company. As competitive baseline algorithms, we use SVMmap [15] and RankBoost [7].

2. SIMILARITY BETWEEN POINT CLOUDS

Learning to rank has a two-level structure. Unlike traditional classification, each instance is not a single feature vector, but a set of feature vectors, each labeled good or bad, associated with a query. We can even train a classifier or ranker for each training query q separately. Given a test query t, we expect that the models corresponding to similar training queries will be more accurate.

In this section we describe how to compute similarity between two queries when a query is represented as a point cloud, i.e., an unordered set of feature vectors[1]. At a high level, our working hypothesis is:

> The proposed similarity measure between queries q, q' should reflect the similarity between the shapes of the point clouds defined by D_q and $D_{q'}$.

Here we study the specifics of how to characterize cloud shapes and their similarities. While we are designing similarity measures between point clouds, we will point out various *invariants* that they do or do not satisfy:

- A basic requirement is that the order of presenting points in a cloud be ignored. This requirement is satisfied by all the similarity measures discussed in this paper.
- For the ranking task, scaling or shifting all points in a cloud should not affect its similarity to other clouds.

2.1 Simple cloud aggregates

One way to compute similarity between point clouds is to represent each cloud as a feature vector by aggregating the feature vectors of individual points in the cloud. In this spirit, we define the following three aggregates:

Mean $\mu \in \mathbb{R}^d$**:** where $\mu = \frac{1}{n_q} \sum_{i=1}^{n_q} x_{qi}$, $n_q = |D_q|$

Variance $\sigma^2 \in \mathbb{R}^d$**:** where $\sigma_j^2 = \frac{1}{n_q} \sum_{i=1}^{n_q} ((x_{qi})_j - \mu_j)^2$: σ^2 is the diagonal elements of covariance matrix of points in the cloud.

Skew $\varphi \in \mathbb{R}^d$**:** where $\varphi_j = \frac{\frac{1}{n_q} \sum_{i=1}^{n_q} ((x_{qi})_j - \mu_j)^3}{\sigma_j^3}$

Once we form the cloud aggregates for two clouds, we can compare the aggregate feature vectors in usual ways, like dot product or Euclidean distance. If the document vectors are in $\mathbb{R}^d$, the time to estimate mean and element-wise standard deviation for query q is $O(dn_q)$, for a total time of $O\left(d \sum_q n_q\right)$. Thereafter, the similarity between any pair of clouds can be done in $O(d)$ time. Evaluating all pairwise similarities will take $O(d|Q|^2)$ time. Although obvious, it is important to note that the mean is not shift- or scale-invariant, and the standard deviation is not scale-invariant.

2.2 Distributional divergence and kernels

2.2.1 KL-Divergence

Another option is to assume that each point cloud has been generated from a multivariate Gaussian, estimate its parameters, and compare the two distributions using those

[1]In this paper we will consider the point clouds as unlabeled, because, for training, we might be able to exploit the point labels, but the test cloud is unlabeled.

estimated parameters. The symmetric KL (Jensen-Shannon) divergence is one measure of distance between distributions that we can use.

Given two point clouds D_1, D_2, let the corresponding mean vector and covariance matrices be $\mu_1, \mu_2, \Sigma_1, \Sigma_2$. Then the KL divergence is given by

$$\mathrm{KL}(D_1\|D_2) = \frac{1}{2}\left(\ln\frac{|\Sigma_2|}{|\Sigma_1|} + \mathrm{tr}(\Sigma_2^{-1}\Sigma_1)\right) + \frac{1}{2}\left((\mu_2-\mu_1)^\top\Sigma_2^{-1}(\mu_2-\mu_1) - d\right) \quad (1)$$

and the symmetric divergence is

$$\mathrm{JS}(D_1, D_2) = \tfrac{1}{2}(\mathrm{KL}(D_1\|D_2) + \mathrm{KL}(D_2\|D_1)). \quad (2)$$

Computing divergence between a pair of queries takes $O(d^3)$ time on account of the inverse and determinant operations. Thereafter, computing divergence between all pairs of queries takes $O(d^3|Q|^2)$. Note that this definition is neither shift nor scale invariant.

2.2.2 Bhattacharyya kernel

Given densities $p_1(x), p_2(x)$ over random variable x, the Bhattacharyya kernel is defined as

$$K(p_1, p_2) = \int_x \sqrt{p_1(x)}\sqrt{p_2(x)}dx. \quad (3)$$

Kondor and Jebara [9] proposed to use the Bhattacharyya kernel between two multivariate Gaussian distributions, whose parameters are estimated from the respective clouds. Fortunately, this can be computed in closed form:

$$\begin{aligned}\Sigma^\dagger &= \left(\tfrac{1}{2}\Sigma_1^{-1} + \tfrac{1}{2}\Sigma_2^{-1}\right)^{-1}\\ \mu^\dagger &= \tfrac{1}{2}\Sigma_1^{-1}\mu_1 + \tfrac{1}{2}\Sigma_2^{-1}\mu_2\\ K(p_1,p_2) &= |\Sigma_1|^{-1/4}|\Sigma_2|^{-1/4}|\Sigma^\dagger|^{1/2}\\ &\quad \exp\left(-\tfrac{1}{4}\mu_1\Sigma_1^{-1}\mu_1 - \tfrac{1}{4}\mu_2\Sigma_2^{-1}\mu_2 + \tfrac{1}{2}\mu^{\dagger\top}\Sigma^\dagger\mu^\dagger\right).\end{aligned} \quad (4)$$

Observe that matrix inversion is needed for each pair of clouds. Like matrix multiplication, this is $O(d^3)$, but the constant is usually somewhat larger. The overall time to evaluate all pairs of cloud distances is $O(d^3|Q|^2)$. Again note that, by design, $K(\cdot,\cdot)$ is sensitive to translation and scaling.

2.3 Our proposal

Let $U = (u_1, \ldots, u_P)$ be the sequence of principal components of D_q, in order of decreasing eigen value. Likewise, let $U' = (u'_1, \ldots, u'_P)$ be the sequence of principal components of $D_{q'}$, in order of decreasing eigen value. Then we define

$$\mathrm{sim}(D_q, D_{q'}) = \frac{1}{P}\sum_{p=1}^{P}\left|u_p^\top u'_p\right|. \quad (5)$$

We take the absolute value because, if u is a principal component of a point cloud, then so is $-u$. Depending on minor numerical perturbation, a PCA program may find either u or $-u$ as a principal component of a point cloud D. Verify, as an extreme case, that $\mathrm{sim}(D, D) = 1$. Also note that

$$\mathrm{sim}(D, D') = \mathrm{sim}(D, D' + \eta),$$

	MAP	PCA based Distance	Symmetric KL Distance	Bhattacharyya Distance	KNN Distance
q2	1.000	0.177	84.584	1.000	95.941
q3	0.887	0.810	48.037	0.917	1.124

Figure 1: Comparison of different similarity measures on synthetic dataset.

where $D' + \eta$ denotes all document vectors in D' translated through vector η; i.e., $\mathrm{sim}(D, D')$ is translation-invariant. Furthermore,

$$\mathrm{sim}(D, D') = \mathrm{sim}(D, \alpha D') \quad \forall\alpha \neq 0,$$

where $\alpha D'$ denotes all document vectors in D' scaled by a factor α; which means that $\mathrm{sim}(D, D')$ is scale-invariant. One potential problem with (5) as a definition of similarity is that clustering algorithms cannot use fast techniques developed for inner-product spaces [6].

The preprocessing step performs PCA on each query cloud, taking $O\left(\sum_q d^2 n_q + \sum_q d^3\right) = O(d^2\sum_q n_q + d^3|Q|)$ time. Thereafter, the time for all pairwise cloud comparisons is $O(d|Q|^2)$. Even though d is just a constant typically between 40–250, in practice, $d|Q|^2 \ll d^3|Q|^2$.

2.4 Illustrative experiments

In order to test the effectiveness of our point cloud similarity measure, we generate synthetic point clouds for three queries q_1, q_2 and q_3. The point cloud of each query q_i has relevant and non-relevant document sets $\{D_i^+\}$ and $\{D_i^-\}$ respectively. Each document in $\{D_i^y\}$ (where $y \in \{+,-\}$) a d dimensional feature vector and generated from a Gaussian distribution $\mathcal{N}(\mu_i^y, \Sigma_i^y)$ with mean μ_i^y and covariance matrix Σ_i^y. We generate the documents while making sure that point cloud of q_1 and q_2 have similar mean while q_1 and q_3 have similar variance.

Now we train a ranking model (SVMMAP) on the documents of query q_1 and test the model on query q_2 and q_3. From the point of view of ranking accuracy we assert that q_1 is more similar to q_2 than q_3, if the ranking accuracy achieved by the model (trained using q_1) is higher on q_2 than q_3. We then compute similarity of query q_1 to queries q_2 and q_3 using the various measures described above. The hypothesis is that a good similarity measure from ranking perspective should find the query with higher ranking accuracy to be more similar to q_1.

Figure 1 shows the result of the above experiment. Query q_1 is more similar to q_2 than q_3 as the model trained on q_1 achieves better MAP score on q_2 than q_3. Also the distance computed using PCA based similarity between q_1 and q_2 is smaller than the distance between q_1 and q_3. But the other similarity measures have a lower distance between q_1 and q_3 compared to that between q_1 and q_2. This shows that our method of computing similarity between two point clouds is more effective than the other methods with regard to ranking.

3. PROPOSED ALGORITHM (LOCALRANK)

3.1 Clairvoyant selection of training query

Our proposed notion of similarity between queries immediately suggests a clustering scheme over training queries.

However, on our way to the final algorithm, we first do some sanity-checking experiments: Is anything to be gained by segregating training queries, as against using all of them in a single model? We begin with an extreme case, with a model w_q per training query $q \in Q$. At test time we choose a model in two different manners.

1. Model w_{q^*} such that $q^* = \arg\max_{q \in Q} \text{sim}(D_q, D_t)$.
2. The *clairvoyant* best model $w_{\tilde{q}}$ where $\tilde{q} \in Q$ such that $w_{\tilde{q}}$ gives the best accuracy for t.

As a baseline we train a single model w_0 using all training queries. Detailed experimental results are presented in Section 4.2.1, but here is a summary:

- Compared to w_0, the accuracy achieved with the clairvoyant choice $\tilde{q}$ is consistently much larger.
- However, the fraction of times that $\tilde{q} = q^*$ is quite small, meaning that the clairvoyant choice is not visible via single query-to-query similarity evaluation.

These preliminary experiments suggest that one-model-per-query over-fragments training data and we can potentially gain accuracy by clustering the training queries. We use complete-link agglomerative clustering in our experiments. Then we train a model on each cluster instead of individual queries. At test time we choose the cluster that contains the most similar training query to the test query t. Pseudocode of our proposed algorithm is shown in Figure 2.

The main experimental question is, will clustering make it any easier to locate the best cluster from which to use the trained model, and whether clustering will smear together queries that are not *quite* similar enough, so as to reduce the efficacy of cluster models for any specific test query. Before we resolve those questions in Section 4, we discuss how to make similarity search more efficient.

Training:
Input: Training queries Q with associated point clouds, number of clusters C, number of principal components P
Output: C clusters and their corresponding models
for each $q \in Q$ **do**
  run PCA(D_q) and record principal components U_q
cluster $\{U_q : q \in Q\}$ into C clusters using complete-link agglomerative clustering with our proposed similarity measure (Equation 5)
for each cluster c **do**
  train a model using queries Q_c belonging to cluster c
  record model w_c obtained by training

Testing:
Input: Set of documents D_t for test query t, C clusters and their corresponding models
Output: Ranked list of documents in D_t
run PCA(D_t) to get principal components U_t
find cluster $c^* = \arg\max_c \max_{q \in Q_c} sim(D_t, D_q)$ where $sim(D_t, D_q)$ is computed using Equation 5
use w_{c^*} to rank D_t

Figure 2: Clustered local learning pseudocode.

3.2 Similarity search using locality sensitive hashing

In learning to rank applications, one can usually be more generous with training time compared to testing time. Any practical proposal for local learning must come up with a fast mechanism to assign a test query to one or few local models compiled during training time. In order to reduce our testing time for selecting the right model for a given test query we make use of locality sensitive hashing (LSH)[2].

Preprocessing
Parameters:
P: Number of principal components (PC) retained for each query
h: Number of hyperplanes in each hashtable, h is tuned
m: Repetition factor for hashtables, m is tuned
Input: Training queries, empty hash-tables $HT[P][m]$
Output: Populated $HT[P][m]$
Data structures: $HT[P][m]$: There are mP hash-tables
for each training query q **do**
  for $p \leftarrow 1$ *to* P **do**
    for $j \leftarrow 1$ *to* m **do**
      insert q in $HT[p][j]$ by hashing the p^{th} PC of q {Note that a different random projection is used for each j}

Testing
Input: $HT[P][m]$ populated with training queries, test query t
Output: Closest training query q as per PCA-based similarity
Data structures: $CQ = \varnothing$: Set of candidate training queries
Initialize two counter arrays $CP[P][|Q|]$ and $CM[P][|Q|]$ to zero {CP for $+u$ and CM for $-u$}
for $p \leftarrow 1$ *to* P **do**
  for $j \leftarrow 1$ *to* m **do**
    hash t into $HT[p][j]$ using its p^{th} PC u_p to get bucket b
    for each train query q in bucket b **do**
      Add q to CQ
      Increment the count $CP[p][q]$
  Repeat the steps 7 to 11 for $-u_p$ i.e. the negative of the p^{th} PC and update the counts in $CM[p][\,]$
for each candidate train query $q \in CQ$ **do**
  Initialize accumulator a_q to 0
  for $p \leftarrow 1$ *to* P **do**
    add max{ $CP[p][q]$, $CM[p][q]$ } to accumulator a_q
Return $\arg\max_q a_q$

Figure 3: Locality Sensitive Hashing pseudocode.

LSH is a method of hashing objects into buckets such that similar objects are hashed into same bucket with high probability and hence speed up the search for similar objects. We use the random projection method of LSH to approximate the cosine distance between the principal components in our PCA-based similarity measure. The basic idea of is to choose a random hyperplane (defined by a random normal unit vector r) at the outset and use the hyperplane to hash input vectors.

Given a principal component u and a hyperplane r, we let $hash(u) = \text{sign}(u^\top r)$. That is, $hash(u) = \pm 1$ depending on which side of the hyperplane u lies. If there are h hyperplanes we get a bit vector $v \in \{0,1\}^h$ of length h by mapping each hash value $\{-1, +1\}$ to $\{0, 1\}$. The bit vector v, interpreted as a number in binary, indexes one of 2^h buckets in the hashtable. A particular bucket will contain

[2] `http://en.wikipedia.org/wiki/Locality_sensitive_hashing`

queries with same bit vector v, i.e., a pair of queries in the same bucket will tend to have higher cosine similarity between the principal components.

We need to adapt LSH in one important way to use it in our application. In our PCA based similarity measure we use multiple principal components of the queries and also the absolute value of the dot product (equation 5). Therefore we need to tailor LSH to work with our similarity measure. We handle multiple principal components using multiple hashtables, one for each principal component. Given that we retain P principal components, this means we use P hashtables. First we populate the hashtables with the training queries using the above described hashing mechanism, once for each principal component. In order to handle the absolute value of the dot product in our PCA-based similarity, we use the principal component u and its negative, $-u$, and generate two separate bit vectors. At test time we retrieve a set of training queries from the hashtables using u and $-u$ and retain the most similar training query. The details of the algorithm are given in the figure 3.

4. EXPERIMENTS

4.1 Experimental testbed

4.1.1 Data sets

We use two data sets to test our ideas. These are described below.

LETOR. For accuracy studies we primarily use the LETOR data set [11], version 3. It consists of seven different data sets (TD2003, TD2004, HP2003, HP2004, NP2003, NP2004, OHSUMED) with a total of 575 queries. We clean the dataset by removing documents of a particular query that has conflicting relevance labels. Cleaning also involves removing those queries that have no relevant document (for details see [3]).

Yandex. Recently, a Russian Internet company called Yandex has released another LETOR-like dataset, as part of the *Internet Mathematics Contest 2009*[3]. Specific permission was obtained from the publishers of the dataset to report results of algorithms discussed in this paper on this dataset. In this dataset, each query-document pair is described by 245 features. Four level relevance judgement is available for 97,290 query-document pairs and can be used for training a model. We clean Yandex dataset in the same way as done for LETOR.

4.1.2 Baseline algorithms and evaluation

We compare our proposed system LocalRank against several baseline algorithms as well as other local learning approaches:

SVMmap: Based on structured learning [14], this directly optimizes a convex upper bound to a loss function that reflects mean average precision (MAP). SVMmap is also the algorithm our system uses on each training cluster.

KNN: This is our implementation of Geng *et al.*'s algorithms [8]. In LETOR datasets (except OHSUMED) we determine the 27 features using the description provided in [8, 13]. For Yandex and OHSUMED we used all the available features to represent the queries as we could not determine the 27 features.

RankBoost: We implemented the standard RankBoost algorithm as in [7].

D&K: This is our reimplementation of Duh and Kirchhoff's system [5]. We implemented ordinary PCA as well as PCA with polynomial and RBF kernels. On our cleaned version of LETOR3, we did not find kernel PCA to improve accuracy beyond ordinary PCA, so we will report on D&K with ordinary "linear" PCA[4].

We report on ranking accuracy in terms of mean average precision and normalized discounted cumulative gain (NDCG) at three ranks: 1, 5, and 10. Since Yandex has non-binary relevance judgement, MAP evaluation measure is not applicable for this dataset.

There are several parameters that need to be tuned across the algorithms. In any of the max-margin methods, the objective is $\frac{1}{2}\|w\|_2^2$ plus a magic parameter[5] B times a suitable sum of slack variables. We pick the best B from the set $\{10,\ 1,\ 0.1,\ 0.01,\ 0.01\}$. In the final results we tune the other parameters in the following way. The number of neighbors, K, in KNN method and the number of clusters, C, in our method are tuned using cross validation. Our method has one more critical parameter, P, the number of principal components to retain while computing the similarity between the point clouds. We observed that generally better result is obtained when P is chosen such that it covers 80% of the variance of the point cloud, that is, the sum of the top P eigen values must reach at least 80% of the sum of all the eigen values.

4.2 Results

4.2.1 Clairvoyant vs. most similar single query selection

Figure 4 compares the accuracies achieved if we use the trained model associated with $\tilde{q}$, the clairvoyant query vs. q^*, the query most similar to the test query. In figure 5 we show where our PCA based similarity ranks the clairvoyant training query. For each test query, we order the training queries by the PCA-based similarity measure and note down the "rank" of the clairvoyant training query $\tilde{q}$. Figure 5 shows for each rank the fraction of test queries for which $\tilde{q}$ was at that rank. We see that

- Compared to the baseline, $\tilde{q}$ achieves substantially better accuracy.
- However, our PCA-based $\text{sim}(q, q')$ is not effective in selecting a query that gives accuracy better than the baseline.
- In fact, the PCA-based similarity can place $\tilde{q}$ at the top in only about 11% of the test queries (Figure 5).

4.2.2 Clustering helps

From figure 4 and 5 we learn that PCA based similarity does not give a direct handle to select the clairvoyant query. Can we save the situation by clustering queries? To answer this we use the proposed agglomerative clustering algorithm with a specified number of clusters.

[3] `http://company.yandex.ru/grant/2009/en/`

[4] To verify our implementation we sent our cleaned data to Kevin Duh. The accuracy he obtained using his code was close to the accuracy we obtained.

[5] Usually called C, but we have already used C to denote the number of clusters.

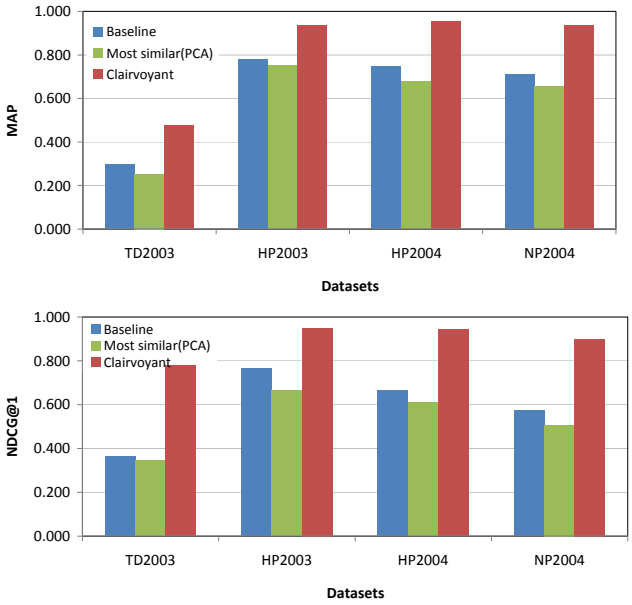

Figure 4: Accuracy obtained using clairvoyant single query $\tilde{q}$ is much better than the baseline. It also beats accuracy obtained using the model corresponding to the most similar (based on PCA-based similarity) training query.

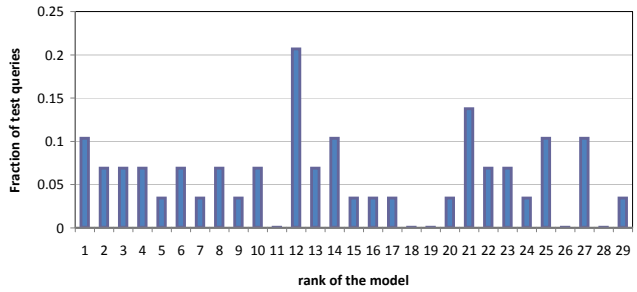

Figure 5: If training queries are ranked by our PCA-based similarity to the test query, the best model to use for the test query is rarely at or even near the very top.

In Figure 6 we plot test MAP (TD2003) as the number of clusters, C, is allowed to increase. We can see that MAP increases from the baseline value obtained by SVMMAP from $C = 1$ to $C = 4$, then drops below the baseline. In this data set there were 29 training queries in each fold. Our plot continues to $C = 29$, which corresponds to no clustering at all (one model per query). As can be seen, no clustering (equivalent to one query per cluster) gives much poorer accuracy than an optimal intermediate level of clustering. Another line in the chart shows the clairvoyant accuracy upper bound, which keeps on improving with increasing C. However, there is no method to find the best cluster except in hindsight; the clairvoyant line just tells us that *some* cluster gets better and better than baseline as C increases. C must be about right to achieve two slightly conflicting goals:

- There should be enough queries in each cluster so that a non-clairvoyant cluster selection algorithm based on similarity has a good chance of picking up a very good model for the test query.
- If the number of clusters is too small, not sufficient query diversity will be recognized, and diverse models will be averaged out, leading to lower accuracy.

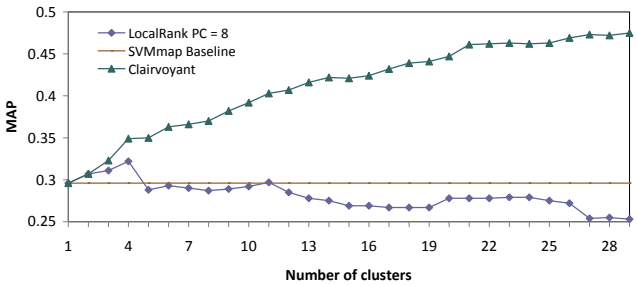

Figure 6: Test MAP vs. C, the number of clusters.

Patterns similar to MAP are seen for NDCG@1 (Figure 7) and NDCG@5 (Figure 8) for another data set. Summarizing the three charts,

- When $C = 1$ our algorithm is the same as the baseline, by definition.
- Peak accuracy occurs uniformly at $C > 1$, suggesting that clustering is clearly useful.
- As C increases further, accuracy decreases, as clusters become too small to build models any better than single-query-based models.

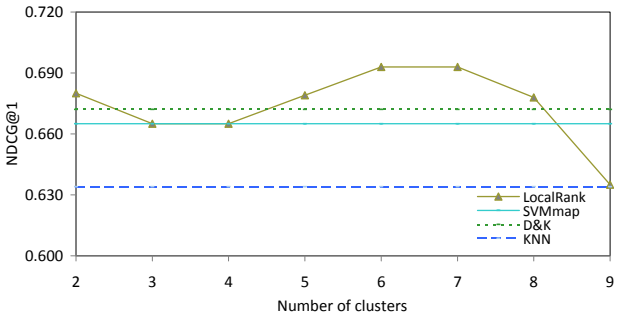

Figure 7: NDCG@1 vs. C for HP2004.

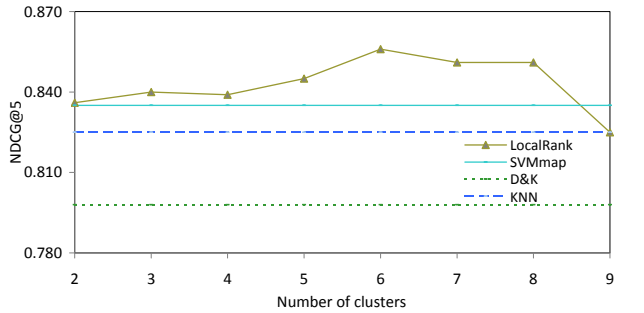

Figure 8: NDCG@5 vs. C for HP2004.

4.2.3 *Comparison with other cloud similarity functions*

Next we ask if the relative sophistication of our PCA-based similarity was indeed required, or would the simple aggregates and moments mentioned in Section 2.1 suffice. Figure 9 shows the results. In this figure, the parameter C (number of clusters) is tuned for individual methods to give the best accuracy.

Clearly the PCA-based notion of cloud similarity adds some value that is missed by simpler moments. Next we

	MAP			NDCG@1			NDCG@5		
	TD2003	HP2003	HP2004	TD2003	HP2003	HP2004	TD2003	HP2003	HP2004
Mean	0.275	0.785	0.765	0.407	0.773	0.681	0.327	0.828	0.855
Variance	0.294	0.777	0.750	0.344	0.751	0.680	0.370	0.823	0.839
Skewness	0.301	0.777	0.760	0.424	0.758	0.694	0.357	0.824	0.827
LocalRank	0.333	0.798	0.769	0.447	0.787	0.707	0.395	0.843	0.859

Figure 9: Comparison of PCA-based cloud similarity with simpler notions of cloud similarity based on simple aggregates.

	MAP			NDCG@1			NDCG@5		
	TD2003	HP2003	HP2004	TD2003	HP2003	HP2004	TD2003	HP2003	HP2004
KL-Divergence	0.313	0.782	0.751	0.427	0.765	0.680	0.371	0.837	0.834
Bhattacharyya	0.265	0.784	0.741	0.367	0.765	0.678	0.331	0.835	0.837
LocalRank	0.333	0.798	0.769	0.447	0.787	0.707	0.395	0.843	0.859

Figure 10: Comparison of PCA-based cloud similarity with other notions of cloud similarity like Bhattacharyya kernel and Symmetric KL divergence.

compare our PCA-based similarity with distributional divergence and point-cloud-kernel based similarity in Figure 10. Here also the parameter C is chosen in the above mentioned way. As visible from Figure 10 and Figure 9 our method consistently outperforms the others. One possible reason could be that these methods are not shift or scale invariant.

4.2.4 *Comparison with other algorithms*

Having studied specific properties and parameters of our approach, we present a broad summary of the accuracy of our algorithm compared against baseline algorithms and other local learning algorithms in the literature.

The accuracies for all the algorithms are shown in Figure 11. We make the following observations from the table:

		TD2003	TD2004	HP2003	HP2004	NP2003	NP2004	OHSUMED	YANDEX
NDCG@1	SVMmap	0.364	0.493	0.765	0.665	0.591	0.573	0.676	0.664
	Rankboost	0.360	0.453	0.714	0.653	0.636	0.530	0.617	0.576
	D&K	0.424	0.480	0.699	0.672	0.636	0.538	0.588	x
	KNN	0.313	0.324	0.762	0.578	0.584	0.499	0.638	0.661
	LocalRank	0.404	0.493	0.765	0.693	0.623	0.530	0.647	0.663
NDCG@5	SVMmap	0.368	0.363	0.829	0.835	0.801	0.830	0.621	0.763
	Rankboost	0.325	0.349	0.854	0.821	0.816	0.768	0.597	0.709
	D&K	0.366	0.378	0.848	0.798	0.836	0.805	0.577	x
	KNN	0.369	0.320	0.817	0.806	0.807	0.770	0.616	0.767
	LocalRank	0.383	0.348	0.837	0.856	0.791	0.828	0.623	0.761
NDCG@10	SVMmap	0.385	0.343	0.840	0.845	0.821	0.847	0.601	0.824
	Rankboost	0.347	0.340	0.868	0.845	0.836	0.806	0.582	0.781
	D&K	0.367	0.350	0.862	0.826	0.860	0.823	0.572	x
	KNN	0.322	0.324	0.831	0.829	0.817	0.836	0.580	0.826
	LocalRank	0.392	0.336	0.850	0.868	0.815	0.847	0.611	0.823
MAP	SVMmap	0.269	0.259	0.781	0.746	0.707	0.709	0.563	x
	Rankboost	0.277	0.263	0.782	0.739	0.740	0.664	0.545	x
	D&K	0.298	0.265	0.770	0.742	0.749	0.678	0.529	x
	KNN	0.232	0.242	0.762	0.714	0.702	0.668	0.547	x
	LocalRank	0.327	0.251	0.789	0.759	0.717	0.687	0.558	x

Figure 11: Summary of MAP and NDCG accuracies for all algorithms and various datasets.

- LocalRank is nearly equal (±0.001) or better than SVMmap 74% of the times.
- LocalRank is better than RankBoost 74% of the times.
- LocalRank is better than KNN 81% of the times.
- LocalRank is better than D&K 64% of the times but we see in Section 4.2.5 that D&K has impractical test time.

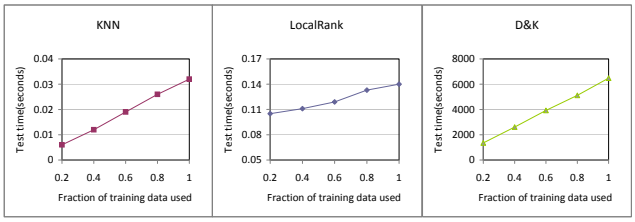

Figure 12: Testing time per query vs percentage of training data used for Yandex.

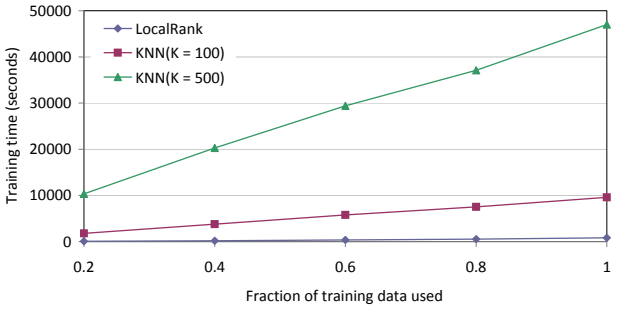

Figure 13: Training time vs percentage of training data used for Yandex.

- Overall LocalRank is the best 42% of the times.

4.2.5 *Running time and scaling behavior*

Our central goal was to match or exceed the benefits from recent local learning algorithms for ranking, while significantly improving upon testing and/or training time. LETOR is a very small data set (100 queries) to perform reliable runtime performance and scalability studies, for which we turned to the Yandex data (7900 queries).

D&K has unacceptably long test times. On Yandex dataset we could not complete a single run of D&K because of its huge size. For just one test query, D&K takes 1.5 hours. That means, to get the results of all the queries D&K will take almost a year.

Figure 12 shows test time per query against the percentage of training data used for Yandex dataset. Note that we choose to use different scales in the Y-axis of the plots to show the scaling effect of increase in training data size for different algorithms. D&K is several orders of magnitude slower than KNN as well as LocalRank. While LocalRank is slower than KNN at test time, KNN's training time is several orders of magnitude greater than LocalRank's. For $K \geq 400$, KNN training time is more than 1.5 days (for five folds), whereas our training time is only 1.1 hours. In Figure 13 we show average training time per fold is plotted against the percentage of training data.

For KNN, the number of models to train is equal to $|Q|$, the number of training queries and that the training time further increases by increasing the size of neighbourhood K of a query. In contrast, the training time increases only moderately for LocalRank on increasing training data, because the number of models to train is equal to the number of *clusters* which is much smaller than $|Q|$.

In order to further improve our testing time we use locality sensitive hashing (LSH) to find the best model to be used for a given test query. The speedup gains from using LSH are visible only if the dataset has a sufficiently large number

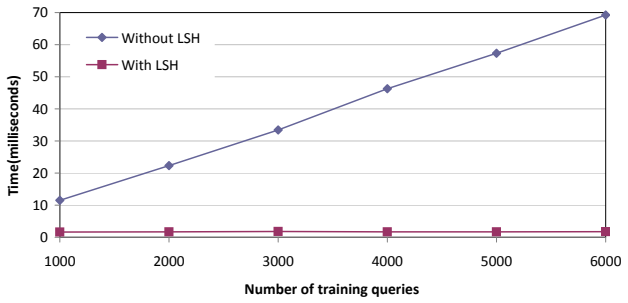

Figure 14: Time taken for selecting model at test-time vs number of training queries for Yandex.

	TD2003	TD2004	HP2003	HP2004	NP2003	NP2004
Quadratic Features	0.298	0.262	0.787	0.749	0.719	0.706
LocalRank	0.327	0.251	0.789	0.759	0.717	0.708

Figure 15: Accuracy (MAP) with non-linear features

of training queries. So we apply LSH on Yandex and plot the time required to find the most similar model to a given test query against the number of training queries. Figure 14 shows that, using LSH, the time required to find the most similar model at test time is almost constant, independent of $|Q|$.

4.3 Non-linearity vs. local learning

Sometimes, local learning may show benefits merely because the hypothesis class in each local model is too weak to properly fit the full data. Many algorithms for learning to rank, including our baseline, SVMmap, uses a linear scoring mechanism. Could it be that using a more complicated non-linear scoring device can obviate the need for local learning altogether? We wanted to explore this question while keeping the rest of our experimental setup unchanged. One way to do this is to use non-linear kernels in max-margin algorithms for learning to rank, such as SVMmap. The problem is that inference becomes intractable.

Therefore we tried to simulate the effect of a non-linear kernel by creating non-linear feature combinations from our raw data. Specifically, in addition to the original features in document vectors, we created new features by multiplying all pairs of original feature values from each feature vector. If the original feature vector was $(x_1, \ldots, x_d)$, then the new feature vector is computed as $[x_1, \ldots, x_d, \ldots, x_i x_j, \ldots, x_1^2, \ldots, x_d^2]$ for $i, j \in 1, \ldots, d$ and $i < j$. Overall this gives $\binom{d}{2} + 2d$ features. Using this encoding has the same effect as using a non-linear polynomial kernel of degree 2. We represent the training and test documents using these non-linear features and evaluate SVMmap. We compare the accuracy of SVMmap, thus modified, against LocalRank with no feature synthesis.

Figure 15 shows the comparison of accuracy numbers. As can be seen, LocalRank outperforms the non-linear feature synthesis approach in terms of MAP accuracy 4 out of 6 times. This seems to indicate that local learning does offer additional predictive quality over and above fitting non-linear but global decision models.

5. CONCLUSION

In this paper we proposed a new representation of queries and a notion of similarity between queries using this representation. We cluster the training queries using this similarity measure and train a model for each individual cluster. At test time, the documents of the query are ranked using the model corresponding to the most similar cluster. We argue that the proposed similarity measure has several desirable properties from ranking perspective, for example, shift and scale invariance. Our experiments show that we are often able to improve the accuracy over baseline and other state of the art local learning approaches. At the same time our algorithm offers reasonable training and test time whereas the other local learning algorithms take unreasonably long time to either train or test.

As future work, we plan to do the following:

- We would like to formulate a point cloud representation that takes the relevance labels of points into account.
- Using queries categorized into known query types, we would like to draw a comparison between query similarity and different categories that the query falls into. This will give us an insight into how the clusters and actual query categories correspond.
- In using single link clustering policy, we had to retain details of each query cloud even if number of clusters was much smaller. We would like to have a compact representation for a cluster of point clouds rather than retaining full information about the clouds constituting a cluster.

6. REFERENCES

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002.

[3] S. Chakrabarti, R. Khanna, U. Sawant, and C. Bhattacharyya. Structured learning for non-smooth ranking losses. In *SIGKDD Conference*, pages 88–96. ACM, 2008.

[4] W. Chu and S. Keerthi. New approaches to support vector ordinal regression. In *ICML*, pages 145–152, 2005.

[5] K. Duh and K. Kirchhoff. Learning to rank with partially-labeled data. In *SIGIR Conference*, pages 251–258. ACM, 2008.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.

[7] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

[8] X. Geng, T.-Y. Liu, T. Qin, A. Arnold, H. Li, and H.-Y. Shum. Query dependent ranking using k-nearest neighbor. In *SIGIR Conference*, pages 115–122. ACM, 2008.

[9] R. I. Kondor and T. Jebara. A kernel between sets of vectors. In *ICML*, pages 361–368, 2003.

[10] P. Li, C. J. C. Burges, and Q. Wu. McRank: Learning to rank using multiple classification and gradient boosting. In *NIPS Conference*, pages 845–852, 2007.

[11] T.-Y. Liu, T. Qin, J. Xu, W. Xiong, and H. Li. LETOR: Benchmark dataset for research on learning to rank for information retrieval. In *LR4IR Workshop*, 2007.

[12] T. Qin, X.-D. Zhang, D.-S. Wang, T.-Y. Liu, W. Lai, and H. Li. Ranking with multiple hyperplanes. In *SIGIR Conference*, pages 279–286. ACM, 2007.

[13] R. Song, J.-R. Wen, S. Shi, G. Xin, T.-Y. Liu, T. Qin, X. Zheng, J. Zhang, G. Xue, and W.-Y. Ma. Microsoft Research Asia at Web Track and Terabyte Track of TREC 2004. In *TREC*, volume 13 of *NIST SP 500-261*, 2004.

[14] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6(Sep):1453–1484, 2005.

[15] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR Conference*, pages 271–278, 2007.