Adaptively Discovering Meaningful Patterns in High-Dimensional Nearest Neighbor Search

Yimin Wu and Aidong Zhang
Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, NY 14260

Abstract

To query high-dimensional databases, similarity search (or k nearest neighbor search) is the most extensively used method. However, since each attribute of high dimensional data records only contains very small amount of information, the distance of two high-dimensional records may not always correctly reflect their similarity. So, a multi-dimensional query may have a k-nearest-neighbor set which only contains few relevant records. To address this issue, we present an adaptive pattern discovery method to search high dimensional data spaces both effectively and efficiently. With our method, the user is allowed to participate in the database search by labeling the returned records as relevant or irrelevant. By using user-labeled data records as training samples, our method employs an adaptive pattern discovery technique to learn the distribution patterns of relevant records in the data space, and drastically reduces irrelevant data records. From the reduced data set, our approach returns the top-k nearest neighbors of the query to the user – this interaction between the user and the DBMS can be repeated multiple times. To achieve the adaptive pattern discovery, we employ a pattern classification algorithm called random forests, which is a machine learning algorithm with proven good performance on many traditional classification problems. By using a novel two-level resampling method, we adapt the original random forests to an interactive algorithm, which achieves noticeable gains in efficiency over the original algorithm. We empirically compare our method with previously well-known related approaches on large-scaled, high-dimensional and real-world data sets, and report promising results of our method.

1 Introduction

Many modern databases, such as multimedia, time series and bioinformatic databases, contain very high dimensional data, where each data record is represented by a multi-dimensional point. To search these databases, similarity search (or nearest neighbor search) is the most extensively used method. Due to its criticalness for searching high dimensional data, nearest neighbor search has been extensively explored, and much sound progress has been achieved. However, most of the existing research results mainly aim to improve the efficiency of multi-dimensional database search. These include multidimensional access method [4, 25, 10, 13], nearest neighbor search and approximation [17, 11], etc.

Recent research results [5, 17, 2] indicate that nearest neighbor search is not always meaningful in high

dimensional data spaces. This is because each attribute of high dimensional data points only contains very small amount of information, and the distance of two points may not always correctly reflect their similarity. So, a multi-dimensional query may have a k-nearest-neighbor set which only contains few relevant points.

Having realized the above deficiency of nearest neighbor search (in high dimensional data spaces), Katayama *et al.* [17] proposed to detect whether the nearest neighbor search of the current query will fail or not, along with its database search. If the probability for the failure (of the nearest neighbor search) is high, they only return the partial search result. To measure the probability of failure, they employ the *distinctiveness* (of the nearest neighbor), which indicates whether the nearest neighbor is located distinctively away from other neighbors or not. Although their approach has been shown to be able to improve the efficiency of nearest neighbor search, it never tries to enhance the search results even if a failure has been detected.

Due to the toughness of automatically locating meaningful nearest neighbors in high dimensional data spaces, Aggarwal [2] presented an interactive database search method, which allows the user to participate in the nearest neighbor search. By using the user's perspective, Aggarwal's approach can effectively examine the neighborhood of the query, and determine discriminatory projects which demonstrate meaningful patterns in the data space.

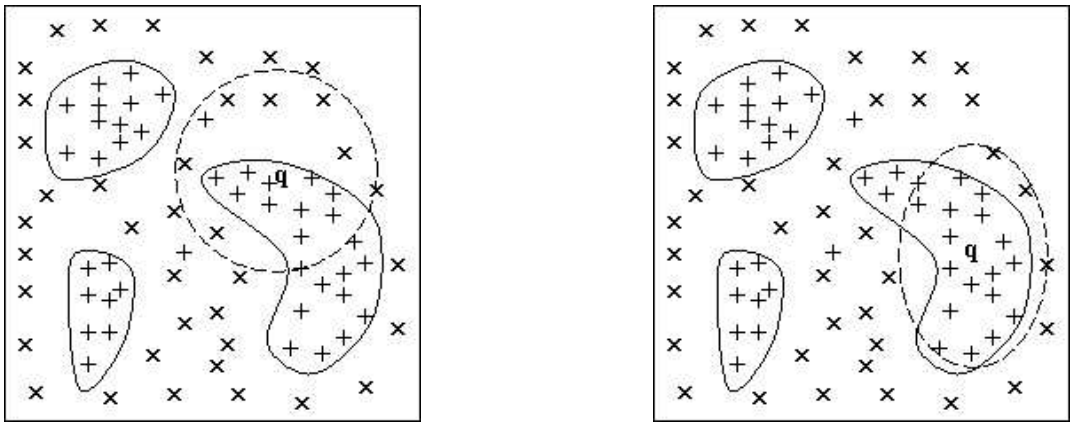
1.1 Related Work

In the past few years, interactive database search (or relevance feedback) has been extensively studied. To address this issue, most existing methods employ query movement and attribute reweighting [22, 27], or query expansion[20, 26]. While most pioneering works perform attribute reweighting using ad-hoc approaches, Ishikawa *et al.* [16] reduced it to an optimization problem, and solved the problem using *Lagrange multiplier*. To further improve search performance, Rui *et al.* [21] extended their approach with a two-level structure, and let the weight matrix switch between a full matrix and a diagonal one to handle small training sets.

To demonstrate the effectiveness and deficiency of the above-mentioned methods, we present (in Figure 1) a typical distribution pattern of relevant points, and the corresponding query results of these approaches. A dominant feature of the distribution pattern is that relevant points tend to distribute nonlinearly in high-dimensional data spaces. For example, in a multimedia database, images of red roses and yellow roses are more likely to form two nonlinear clusters in the color space, while all rose images may be considered as relevant by the user. The nonlinear distribution of relevant points make it extremely difficult to search meaningful nearest neighbors in high dimensional data spaces.

Figure 1 (a) demonstrates that the primitive nearest neighbor search generates a circled boundary [1] in

[1]To simplify our discussion, we assume that the distance metric is the most extensively used Euclidean distance (or L_2 norm



(a) Primitive NN search (b) Traditional relevance feedbacks

Figure 1: An illustration of distribution pattern of relevant points, where "+" denotes relevant points and "$\times$" denotes irrelevant ones.

the data space, and its performance is usually poor, unless the query is located near the center of a large cluster of relevant points. Figure 1 (b) shows that, by employing query movement and attribute reweighting techniques, most traditional relevance feedback approaches [22, 21] can move the query toward the centroid of a relevant cluster, and generate an elliptic hyper-surface to cover relevant points better. However, the common drawback of these approaches is that they are prone to be trapped by local optimum, especially for highly selective queries. This is because each relevant cluster is wrapped by irrelevant points, which hinder the above-mentioned approaches from exploring multiple relevant clusters. So, these approaches often try to adjust their queries to cover one relevant cluster as perfect as possible.

Query-expansion-based approaches, such as those proposed in [20, 26], can find relevant points from multiple clusters. However, the query-expansion-based methods can not achieve good performance unless users can provide multiple query points (with at least one from each relevant cluster) in the beginning. Otherwise, with the neighborhood of each relevant cluster saturated with irrelevant points, the query-expansion-based methods are still vulnerable to local optimum. Porkaew *et al.* [20] demonstrated that query expansion only outperforms query movement marginally.

1.2 Our Contributions

In this paper, we present a novel *adaptive pattern discovery* method to search high dimensional databases. To improve search performance, we allow the user to participate in the database search by labeling (part of) returned points as relevant or irrelevant. By using user-labeled data points as training samples, our method

distance). However, the discussion in this section is applicable to any L_p ($p = 1, 2, ..., \infty$) norm distance.

adaptively discovers the meaningful distribution patterns of relevant points, and employs these patterns to improve the quality of high-dimensional nearest neighbor search. To accomplish *adaptive pattern discovery*, we train a classifier to learn the meaningful distribution patterns of relevant points from online training samples. With the classifier, we perform a data reduction in the data space by filtering all points classified as irrelevant. From the reduced data set, we return the k nearest neighbors of the query to the user. By counting out those points that are classified as irrelevant, our method dramatically reduces the number of candidate points that can be considered as nearest neighbors. The above data reduction can effectively improve the *distinctiveness* [17] of nearest neighbors, and enhance the chance to obtain *meaningful* [2] search results.

The classifier used in our system is the *random forests* [9], which is a machine learning algorithm with proven good performance on many traditional classification problems. By using a novel two-level resampling method, we adapt the original random forests to an interactive algorithm, which achieves noticeable gains in efficiency over the original algorithm. Extensive experiments (cf. Section 5) on large-scaled, high dimensional and real-world data sets indicate that our method outperforms previously well-known related approaches [21, 16] by at least 20%.

The rest of this paper is organized as follows. Section 2 introduces some useful notations and our distance metric. Our adaptive pattern discovery method are then presented in section 3. Section 4 specifies the experimental setup. Our empirical results are presented in Section 5. Finally, we conclude in Section 6.

2 Useful Notation and Distance Metric

2.1 Useful Notation

In this section, we introduce some useful notations which will be used in the rest of this paper.

Let the data space be $F = \{f_1, ..., f_M\}$, where M is the dimensions of F. We denote the database by $db = \{t_1, ..., t_I\}$, where I is the size of db. Each database record $t_i \in db$ is then represented by a real-valued vector (i.e., point) $\vec{t_i} = [t_{i,1}, ..., t_{i,M}]^T$, where $\vec{t_i}$ is an instantiation of the data space F. Similarly, the query q is represented by vector $\vec{q} = [q_1, ..., q_M]^T$. To calculate the distance between data record t_i and query q, we use the weighted Euclidean distance presented in Section 2.2.

Before searching nearest neighbors of the query, we first train a random forest h to classify each database record t_i into one of the two classes: relevant (denoted by 1) and irrelevant (denoted by 0). To train the random forest, we obtain a training set $S = \{(s_1, v_1)..., (s_N, v_N)\}$ from the user, where N is the size of the training set. For each training instance $(s_n, v_n) \in S$, s_n is a user-labeled point represented by $\vec{s_n} = [s_{n,1}, ..., s_{n,M}]^T$, where $v_n \in \{0, 1\}$ is its class value. We denote the set of all relevant points as U, and the set of all irrelevant points as R, with $S = U \cup R$.

Sometimes, it is necessary to distinguish the class assigned by the random forest (to a point) from the class provided by the user. When confusion arises, we call a point as classified-relevant/-irrelevant if it is classified by the random forest as relevant/irrelevant, and use the term relevant/irrelevant for user-labeled points only. We term the set of classified-relevant/-irrelevant points as classified-relevant/-irrelevant set.

During database search, one interaction process (between the user and the DBMS), which includes the user labeling currently returned points and the DBMS refining the previous search results accordingly, is often called an *iteration*.

2.2 Distance Metric

In this section, we present our approach for calculating the distance between the query and the database points. To calculate the distance between query q and the data point t, we use the following weighted Euclidean distance metric:

$$d(q,t) = \sum_{m=1}^{M} w_m \times (q_m - t_m)^2, \tag{1}$$

where w_m specifies the importance of attribute m.

To calculate the weights of attributes, our method [27] employs both relevant points and irrelevant ones from the training set. Before presenting our attribute re-weighting method, we first give the following definitions:

Definition 1 *On the axis of attribute m, we define* dominant range G_m *as the range spanned by the relevant points, as shown in Figure 2. We then define the* discriminative factor δ_m *of attribute m as:*

$$\delta_m = \frac{|U_m^\delta|}{|U|}, \tag{2}$$

where U is the irrelevant point set and $U_m^\delta \subseteq U$ is the subset of U falling outside of G_m.

The *discriminative factor* δ_m indicates the ability of attribute m in discriminating relevant points from irrelevant ones.

Denote the relevant point set by $R = \{r_1, ..., r_T\}$, the weight w_m is updated by:

$$w_m = \frac{\delta_m}{\sigma_m^R}, \tag{3}$$

where σ_m^R is the standard deviation of the sequence $R_m = [r_{1,m}, ..., r_{T,m}]$, which is obtained by stacking the mth attribute values of relevant points. Unlike Mars approach [22], which only uses σ_m^R calculated from relevant points, our approach combines relevant points and irrelevant ones to re-weight attributes.

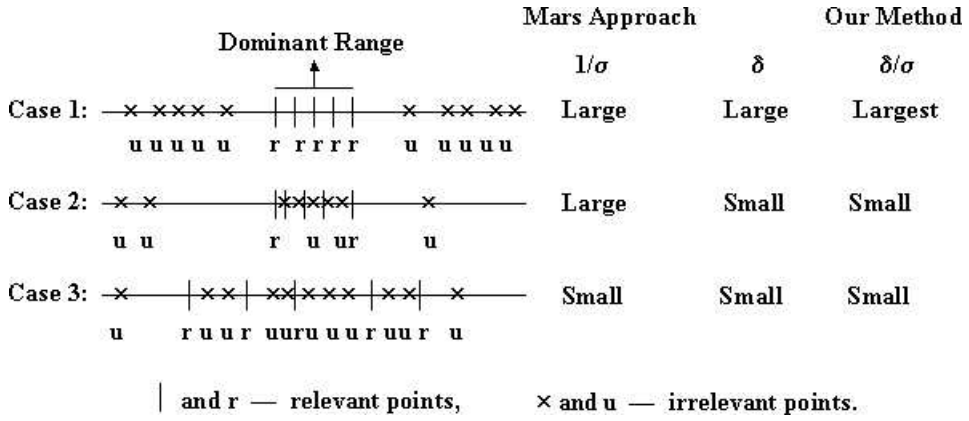


Figure 2: The weight value of one attribute in different cases.

Our method and Mars approach are compared in Figure 2. Mars approach assigns large weights to all attributes that allocate relevant points together. However, due to the complexity of high-dimensional data space, many relevant points in the database may share the same irrelevant attributes, which cluster many irrelevant points and relevant ones together. By inappropriately assigning large weights to irrelevant attributes (as indicated by case 2 in Figure 2), Mars approach may bring in more irrelevant points in the next iteration and make the attribute weights trapped in some sub-optimal state. On the contrary, our approach employs relevant points as well as irrelevant ones to calculate attribute weights. And hence, our approach correctly distinguishes case 1 from case 2, and only assigns large weights to the former.

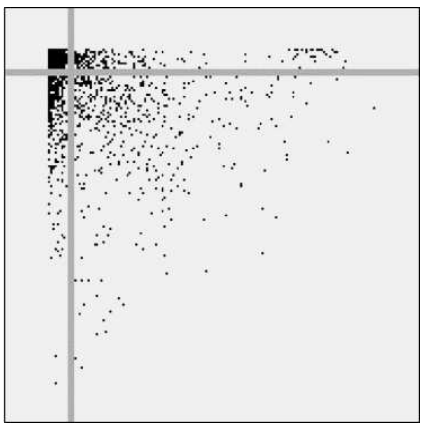
3 Adaptive Pattern Discovery

3.1 Criticalness for Pattern Discovery

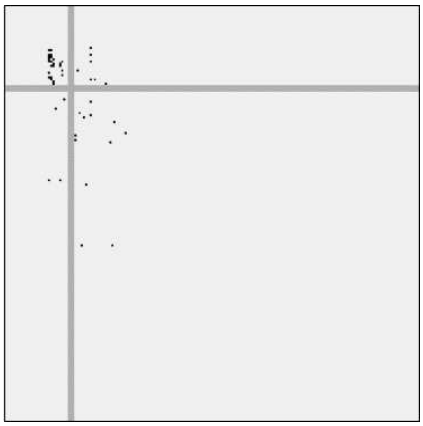
To demonstrate the criticalness of pattern discovery in high-dimensional nearest neighbor search, we present the distribution patterns of a high-dimensional data set, which contains 4,126 data points with 32 features. In the data set, each data point represents a COREL image with a 32-feature color histogram [18].

After examining many 2D projections of the 4,126 data points, we present several typical distribution patterns in Figure 3. Figure 3 (a) indicates that some features (such as features 3 and 11) exist correlation to some extent. However, due to the sparseness of high-dimensional data spaces, most features are independent from each other, as shown in Figure 3 (c). Figure 3 (b) and (d) show the distribution patterns of 75 eagle images in the data space. If one of these images is used as the query, these 75 points are relevant points that are supposed to be found. Figure 4 presents 12 of these 75 eagle images.

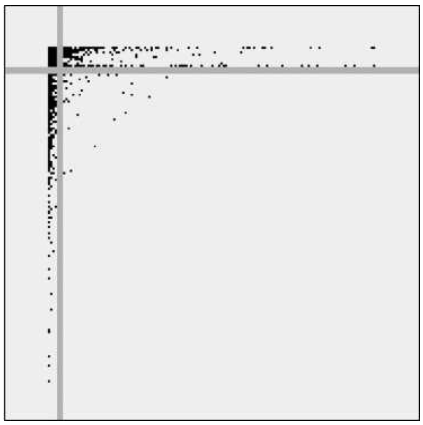
From Figure 3, we can make the following observations:



(a) features 11,3



(b) features 11,3 of eagles



(c) features 15,3

(d) features 15,3 of eagles

Figure 3: 2D projections for the data set. (a) and (c) illustrate the 2D projects of all 4,126 points, while (b) and (d) demonstrate the 2D projects of 75 points representing eagle images. The vertical axis always corresponds to the 3rd feature, while the horizontal axis corresponds to the 11th feature in (a),(b), and corresponds to the 15th feature in (c), (d).

- Since relevant points usually scatter sparsely in the high-dimensional data spaces, the neighborhood of each relevant point inevitably contains (more or less) irrelevant points. This explains why the primitive nearest neighbor search fails so often in high dimensional data spaces [1, 2].
- Figure 3 (d) indicates that, if projected onto the axis of feature 15, most relevant points belong to one of two separate clusters. This confirms recent research results [15, 2] that there may exist meaningful distribution patterns in low-dimensional projection of the high-dimensional data.
- Due to the nonlinearity of the presented meaningful pattern, traditional feature re-weighting approaches [22, 21], which assume relevant points distribute linearly, are unlikely to achieve satisfactory performance on queries of eagle images.

Figure 4: Some examples of eagle images.

- By discovering the meaningful distribution patterns like the one presented in Figure 3 (d), we can employ them to filter most irrelevant points. This filtering operation can effectively reduce irrelevant points in the neighborhood of the query, and dramatically improve the meaningfulness [2] of resulting nearest neighbors.

3.2 Introduction of the Random Forests Algorithm

In this section, we briefly introduce the random forests algorithm [9], which trains a composite classifier from an ensemble of tree classifiers to improve classification accuracy. To get the composite classifier, it combines the bagging [7] technique with random feature selection [9], and achieves favorable performance over the state-of-art methods for combining multiple classifiers, such as bagging and Adaboost [14].

The tree classifier trained in the random forests is the classification and regression tree (CART) [8]. To allocate training data to two children nodes, CART searches through all features to find the best split at each node. If the current node only contains training data of one specific class, CART makes it a leaf node. This leaf node then assigns all test data falling into it to that class. All test instances are run down the tree to get their predicted classes. Since an feature can be used to split multiple times, the CART is able to find nonlinear clusters of relevant points.

We denote the random forest by $h = \{h_j(\vec{t}, \theta_j), j = 1, ...J\}$, where $h_j \in h$ is the jth tree classifier. In the random forest h, each tree classifier h_j is trained with a bootstrap [7, 12] sample $S_j \subset S$ and a random vector θ_j, where J is the number of classifiers in h. The bootstrap sample S_j is generated by randomly resampling the original training set S without replacement, while the random vector θ_j is generated, independent of the past random vectors θ_1,..., θ_{j-1}, but with the same distribution. When growing h_j, a best

split is chosen as the best one from M' randomly selected features on each node. As shown by Breiman [9], a random forest is insensitive to M', the number of features to select at each node, and performs optimally (in general) when M' is set to the square root of M (i.e., the dimensions of the data space F).

To classify input point $\vec{t}$, random forest h lets its member classifiers vote for the most popular class, where each classifier casts a unit vote. If enough trees can be trained, the random forest will not overfit, and only generate a limited value of generalization errors [9].

3.3 Biased Random Sample Reduction

For large sample sets, the computational cost for training the random forest can become prohibitively high. For example, in our experiments, it may need more than 30 seconds to train a reliable random forest from a sample set with 800 training samples (cf. Section 5.4). To address this issue, we propose a novel two-level resampling technique to improve the efficiency of random forests.

To improve the efficiency of the random forest, we first present an analysis of its computational cost, which is dominated by training its member classifiers (CART). If each CART is grown to a uniform depth D (i.e., to 2^D terminal nodes), then the computational cost for training the CART is proportional to $N' * (D+1) * \log N'$ [8], where N' is the size of the bootstrap sample set. So, the computational cost $T(h)$ of random forest h is calculated by:

$$T(h) \propto J * N' * (D+1) * \log N' \tag{4}$$

where J is the number of tree classifiers.

Since D is also related to N', $T(h)$ can be considered as a function of $N' * J$, and is adjustable by choosing appropriate values for N' and J. However, if J is too small, the performance of random forests will degrade dramatically due to the overfitting problem (see Section 5.1). So, J is often set to some constant value. Since N, the size of S, usually increases with the number of iterations, the cost of the random forest can be reduced by setting N' to be no larger than a specific empirical threshold N_0. As introduced in the last section, the random forests train each tree classifier h_j with a bootstrap sample $S_j \subset S$, whose size is about $0.63 * N$ [12]. To ensure $N' \leq N_0$, we propose a two-level resampling approach, called *Biased Random Sample Reduction (BRSR)*, to reduce the size of the sample set for each tree classifier.

Figure 5 illustrates our $BRSR$ resampling method. Unlike the original random forests, which train each classifier h_j directly with a bootstrap sample S_j, our BRSR approach generates a series of *reduced sample sets* $B = \{B_g | g = 1, ..., G, \ and \ B_g \subset S\}$, where G is the number of reduced sample sets. From each reduced sample set B_g, our method produces $P = J/G$ [2] bootstrap samples $S_g = \{S_{g,p}, p = 1, ..., P\}$, with

[2] Without losing generality, we assume J is always divisible by G.

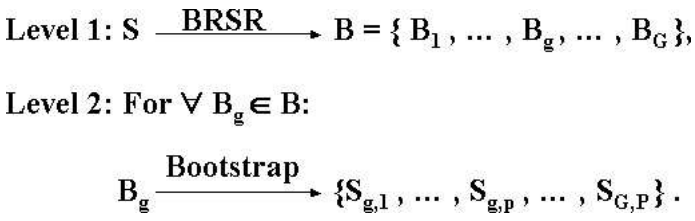


Figure 5: An illustration of our resampling method.

the restriction that the size of each sample set $S_{g,p}$ is no larger than N_0.

To present $BRSR$ in detail, we need to coin several useful definitions. At first, we define the random sample reduction (RSR) approach as follows:

Definition 2 *Random sample reduction (RSR) generates, from the given sample set S, G reduced sample sets $B = \{B_g | g = 1, ..., G, \ and \ B_g \subseteq S\}$, with $\bigcup_{g=1}^{G} B_g = B$; $B_g \cap B_x = \phi, for \ \forall g \neq x$; and $|B_g| = |S|/G$. To generate B, we might randomly permute S, and divide the permuted set into G non-overlapped and equally sized subsets.*

However, during high-dimensional database search, the size of the relevant set R is often smaller than the size of the irrelevant set U. So, it is often favorable to reduce more irrelevant points than relevant ones. This leads to the following definitions:

Definition 3 *Relevant-bootstrap irrelevant-RSR (RBIRSR) generates the reduced sample sets as $B = \{B_g = B_g^R \cup B_g^U, g = 1, ..., G\}$, where B_g^R is a bootstrap sample of the relevant set R, and B_g^U is the gth RSR set generated form irrelevant set U. The size of each reduced sample set can then be calculated by*

$$\alpha = 0.63 * |R| + |U|/G, \tag{5}$$

where $G \geq 2$. We call α the $RBIRSR$ sample size.

Based on the above definitions, we define our BRSR method as follows:

Definition 4 *Biased random sample reduction (BRSR) employs $RBIRSR$ to generate G reduced sample sets B if the $RBIRSR$ sample size $\alpha \leq N_0$. Otherwise, it uses the RSR to generate the reduced sample sets.*

Whenever possible, $BRSR$ reduces more irrelevant points than it does to relevant ones. This is why it is called a *biased sample reduction* method.

Algorithm 1 demonstrates how to calculate the value of G, and how to choose the appropriate method to obtain the reduced sample sets. The following are three cases of the algorithm: First, directly apply the

Algorithm 1 *BRSR: Biased Random Sample Reduction*

Input 1) training set S; 2) relevant point set R; 3) irrelevant point set U.
Initialize 1) $G \Leftarrow 1$; 2) $B \Leftarrow \{S\}$; 3) Resampling method $\mathcal{R} \Leftarrow$ "$Bootstrap''$.
if $N > N_0$ **then**
  if $N \leq 1.5 * N$ **then**
    $G \Leftarrow 2$;
  else
    $G \Leftarrow \lceil N/N_0 \rceil$;
    $\alpha \Leftarrow 0.63 * |R| + |U|/G$;
    if $\alpha \leq N_0$ **then**
      $\mathcal{R} \Leftarrow$ "$RBIRSR''$;
    else
      $\mathcal{R} \Leftarrow$ "RSR'';
    end if
  end if
end if
Generate G reduced sample sets in B with the resampling method specified by $\mathcal{R}$.
Output: the reduced sample sets B and its size G.

original random forests when $N \leq N_0$. Second, set $G = 2$, and generate two reduced sample sets using bootstrap, if $N \in (N_0, 1.5 * N]$. Third, set $G = \lceil N/N_0 \rceil$, and generate G reduced sample sets using $BRSR$ for all $N > 1.5 * N$. Since the size of bootstrap sample is 63% of the original sample set [12], it is easy to verify that the size of each sample set $S_{g,p}$ is no larger than N_0 in all cases.

3.4 Adaptive Pattern Discovery

With the *BRSR* method presented in Algorithm 1, we can adapt the original random forest (ORF) to the *interactive random forests (IRF)*, which runs more efficiently than ORF. Similar to the original random forests, IRF also trains a tree classifier $h_{g,p}$ with each sample set $S_{g,p}$ and a random vector $\theta_{g,p}$. And hence, our interactive random forests $h^* = \{h_{g,p}, g = 1, ..., G, p = 1, ..., P\}$ contains $J = G * P$ tree classifiers, and it lets all the member classifiers vote when classifying input data points.

The *interactive random forests (IRF)* is presented in Algorithm 2. By setting N_0 to some large value, IRF is equivalent to ORF. So, IRF can be considered as a generalization of ORF, since the latter is a special case of the former.

By employing $BRSR$, our interactive random forests accommodate the capability to adjust its computational cost according to the system performance. As to the effectiveness, $BRSR$ may have double-bladed effect on the classification accuracy. On the one hand, it may harm the classification accuracy by worsening the overfitting problem, since less trees are trained from each reduced sample set B_g [9]. On the other hand,

Algorithm 2 *Interactive Random Forests*

Input 1) training set $S = \{(s_1, v_1)..., (s_N, v_N)\}$,where $v_n = 1$ or 0;2) data space $F = \{f_1, ..., f_M\}$;3) J: the number of tree classifiers to grow;4) N_0: threshold of sample set size.
Initialize 1) set the interactive random forests $h^* \Leftarrow \phi$.
Call Algorithm 1 to obtain the G reduced sample sets B;
$P \Leftarrow J/G$;
for $g = 1$ to G **do**
  for $p = 1$ to P **do**
    generate a bootstrap sample $G_{g,p}$ from $B_g \in B$;
    train a tree classifier $h_{g,p}$ with $G_{g,p}$;
    $h^* \Leftarrow h^* \cup \{h_{g,p}\}$;
  end for
end for
Output: the interactive random forests:

$$h^*(\vec{t}) = \begin{cases} 1 & \text{if } \sum_{g=1}^{G} \sum_{p=1}^{P} h_{g,p}(\vec{t}) \geq \frac{J}{2}, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

it can improve the performance by reducing the correlation of generalization errors among tree classifiers, which usually improves classification accuracy of the random forest [9]. Experimental results (cf. Section 5.4) indicate that IRF can improve the efficiency by 2 to 3 times, while only causes trivial degradation on search quality.

During database search, the interactive random forests trained with Algorithm 2 might output less than k classified-relevant points. To address this issue, we define the *relevance probability* of record t as follows

Definition 5 *The relevance probability $P(1|\vec{t})$ of record t is the the number of classifiers that classify t as relevant over the total number of classifiers. Its formal definition is given by*

$$P(1|\vec{t}) = \frac{\sum_{g=1}^{G} \sum_{p=1}^{P} h_{g,p}(\vec{t})}{J}. \tag{7}$$

The larger is $P(1|\vec{t})$, the more confident it is to output t as relevant. So, our method returns some classified-irrelevant records with the largest $P(1|\vec{t})$ values, in case less than k records were classified as relevant.

For each query q, our method runs an initial database search, and returns the k nearest neighbors of q to the user. It then ask the user to provide the initial training sample set S by labeling (part of) returned records as relevant or irrelevant. Our method then iteratively runs the *High-Dimensional Nearest Neighbor Search* algorithm, presented in Algorithm 3, to improve the initial search results.

Figure 6 (a) demonstrates a typical query result of Algorithm 3. It indicates that our approach, by applying *adaptive pattern discovery*, can effectively overcome the nonlinear distribution of relevant points, and locate relevant points from multiple nonlinear relevant clusters.

Algorithm 3 *High-Dimensional Nearest Neighbor Search*

Input 1) training set $S = \{(s_1, v_1)..., (s_N, v_N)\}$,where $v_n = 1$ or 0; 2) q: the query point; 3) the data space: $F = \{f_1, ..., f_M\}$; 4) J: the number of tree classifiers to grow; 4) N_0: threshold for the size of training sample set; 5)W: the maximum iteration times; 6)k: the number of nearest neighbors returned to the user.

1: **for** $w = 1$ to W **do**
2: **Call** Algorithm 2 to train an interactive random forest h^*.
3: **Classify** database points to get the classified-relevant set Γ.
4: **if** $|\Gamma| < k$ **then**
5: **Find** $k - |\Gamma|$ classified-irrelevant points with the largest relevance probability, and add them into Γ.
6: **end if**
7: **Find** the k nearest neighbors of q from Γ, and return them to the user.
8: **Obtain** the new training samples from the user, and merge them with S.
9: **Update** feature weights according to Formula 3.
10: **Update** query q by setting it to the centroid of all relevant points, that is, set the the features of q to the average value of all relevant points.
11: **end for**

3.5 Scalability of Our Method

In Algorithm 3, we use the random forest to classify all the database points. This may be too time-consuming for very large-scaled databases (such as those with millions of records). To address this issue, we employ a sampling-based query estimation method to improve search efficiency.

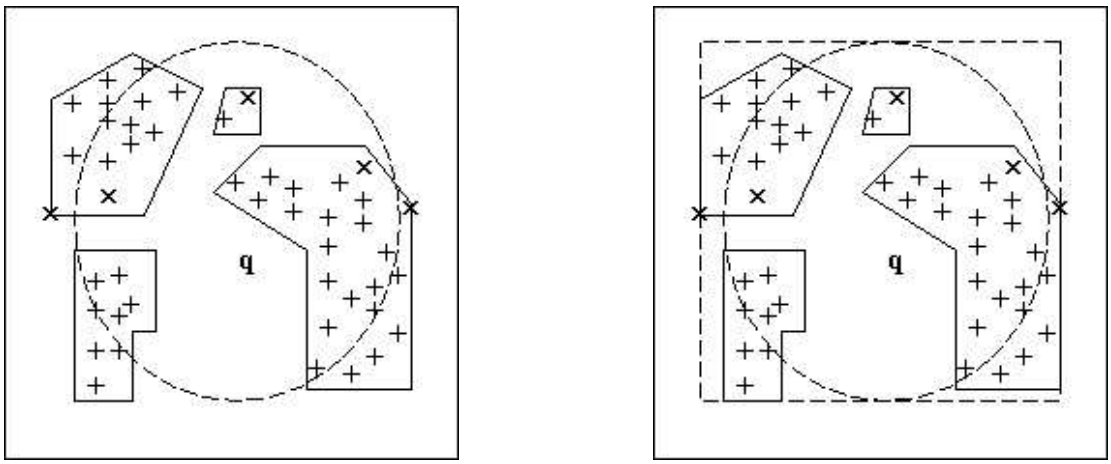
The query estimation method we used is the one proposed by Chen *et al.* [11]. To search the k nearest neighbors of query q, they propose finding $\lambda * k$ sampled nearest neighbors from a sample set $db' \subset db$, where db is the database and $\lambda = |db'|/|db|$ is the sample rate. To approximate the nearest neighbors, they present several mapping strategies to calculate a hyper-square/-rectangle in the data space, which covers all the $\lambda * k$ sampled nearest neighbors. From all points (in db) covered by the calculated hyper-square/-rectangle, the top k nearest neighbors (of the query) are returned to the user.

The above-calculated hyper-square/-rectangle can be considered as a range query, which is formally defined by Chen *et al.* [11] as follows:

Definition 6 *A range query Q is a conjunction of M range predicates, i.e., $Q = \beta_1 \wedge \beta_2 \wedge ... \wedge \beta_M$, where $\beta_i = \gamma_i \leq f_i \leq \delta_i$ is a range query on the ith feature. The result of the range query Q, denoted by Q(db), is the set of points from db that satisfy Q. That is:*

$$Q(db) = \{t \in db | for\ \forall i \in [1, M],\ \gamma_i \leq t_i \leq \delta_i\}. \tag{8}$$

The advantage of using the above range query is that multiple B-trees, with one B-tree for each feature,



(a) Query result of Algorithm 3 (b) Query result of sample-based query (see Section 3.5)

Figure 6: An illustration of our query results.

can be used to support nearest neighbor search. With this indexing approach, $Q(db)$ can be calculated as the intersection of the range query results on all features. This indexing structure has been integrated in *Oracle 8i*, and was shown to perform well on indexing high-dimensional data spaces [13]. Other index structures, such as the VA-File [25], etc., can also be used to support the evaluation of range query in high-dimensional data spaces. In very high-dimensional data spaces, the above mentioned indexing methods outperform many existing multidimensional indexing structures (such as $R^*\text{-}tree$ [3] and $X\text{-}tree$ [4], etc.), which usually perform worse than a linear search if the dimension is higher than 15 to 20.

To integrate the above query estimation method, we replace the steps 3 to 7 of Algorithm 3 with the following steps:

- **Classify** all records in the sampled data set db' to get the sampled classified-relevant set Γ'.
- **If** $|\Gamma'| < \lambda * k$, from db', find $\lambda * k - |\Gamma'|$ classified-irrelevant records with the largest relevance probability, and add them into Γ'.
- **Find** the $\lambda * k$ sampled nearest neighbors of q from Γ', and map the sampled query result to a range query Q using the $MBSS$ [11] query mapping strategy.
- **Evaluate** the range query result $Q(db)$ with the support of multiple B-trees.
- **Classify** all records in $Q(db)$ to get the classified-relevant set Γ.
- **If** $|\Gamma| < k$, from $Q(db)$, find $k - |\Gamma|$ classified-irrelevant records with the largest relevance probability, and add them into Γ.

- **Find** the k nearest neighbors of q from Γ, and return them to the user.

Figure 6 (b) presents the query result when the above sampling-based query estimation method is used. After finding the $\lambda * k$ sampled nearest neighbors from the sampled data set db', we employ the $MBSS$ mapping strategy to obtain the range query Q, which is demonstrated by the dashed square in Figure 6 (b). With the range query Q and appropriate indexing structure, we can efficiently evaluate the corresponding query result $Q(db)$. On obtaining $Q(db)$, our method classify all points from $Q(db) \subset db$, and find the k nearest neighbors from classified-relevant records. Chen *et al.* [11] show that the sampling-based query estimation methods (such as $MBSS$) can maintain high recall, while improving the search efficiency dramatically.

4 Experimental Setup

4.1 Data Sets

The following two real-world data sets were used to test the presented method:

- **PENDIGITS:** This data set [3], titled PENDIGITS, is obtained from the UCI machine-learning archive. The PENDIGITS data set contains 10,993 data points. Each data point represents a hand-written digit with 16 attributes, which are spatially resampled coordinates. In addition to its attribute values, every data point also has a class value between 0 and 9.

- **COREL:** This data set contains 31,438 data points with 179-attributes, where each point represents a color image using five image features: the first one is a 64-attribute color coherence vector [19] in the HSV color space; the second one is a 9-attribute color moments [18] extracted from the L*a*b color space; the third one is a 10-attribute wavelet-based texture descriptor [22]; the fourth one is a 64-attribute edge coherence histogram [6]; and the fifth one is a 32-attribute Fourier shape descriptor[6]. We concatenate the above five features to get a 179-attribute vector for representing the images.

To search the *PENDIGITS* data set, all data points with the class value of 1 are labeled as relevant, and all other points are labeled as irrelevant. The *query* set consists of all the 1,116 positive points.

To search the COREL data set, we use a query set with 5,172 data points, which represent images from 44 semantic categories (such as rose, butterfly, tiger, eagle, penguin, falls and city, etc.). To evaluate the search results, only images from the same semantic category of the query are considered as relevant.

[3] URL of PENDIGITS: *ftp://ftp.ics.uci.edu/pub/machine-learning-databases/pendigits/.*

4.2 Performance Evaluation

Precision and *recall* are used to evaluate the performance of database search. *Precision* is the number of the returned relevant points over the total number of returned points, and *recall* is the number of the returned relevant points over the total number of relevant points in the database. The average *precision* and *recall* of all queries are used as the overall performance.

During interactive database search, users are required to label a fixed number of nearest neighbors as relevant or irrelevant. This number is often called the *scope*. Since the database search performance also varies with scope, we conducted experiments on scopes of 80 and 120 for PENDIGITS queries, and tested COREL queries for scopes of 20, 40, 80 and 120.

For a comparison, we provide the performance of the OPL method [21] under the same experimental conditions. OPL is an extension of the approach proposed in *MindReader* [16], and has been extensively used as a base line for performance comparison [23, 24].

4.3 Thresholds

We test the performance of our method when N_0, the threshold for the size of sample set, equals to 240 and $1,200$, respectively. To distinguish these two cases, we use IRF to denote the former, and use ORF to identify the latter. By setting N_0 to 240, IRF achieves an echo time of less than 10 seconds on the state of art machine (see Section 5.4), while performing almost optimally (see Section 5.3). In our experiments, the interactive random forest is equivalent to the original random forest when $N_0 = 1,200$. This is because we run no more than 10 iterations for each query, and the maximum number of nearest neighbors labeled in each iteration is 120.

5 Empirical Results

5.1 Tree number to grow

Although Breiman [9] suggests training as many trees as possible, Formula 4 shows that the computational cost of the random forest increases linearly with the number of trees. So, we need to empirically minimize the number of trees, while maintaining the performance close to optimal.

Figure 7 demonstrates the precision achieved by our method on COREL queries when the scope is 20. Obviously, a random forest with 60 trees dramatically outperforms a random forest with 30 trees. This is because the overfitting problem is rather severe for a random forest with an inadequate number of trees [9]. After 10 iterations are run, a random forest with 60 trees outperforms a forest with 30 trees by 20%.

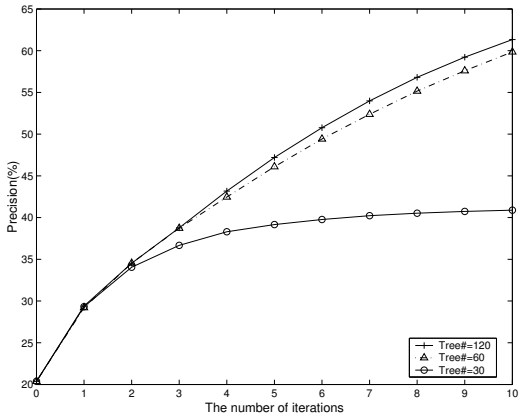


Figure 7: Precisions achieved when 30, 60 and 120 trees are grown. Each curve indicates how the precision increases with the number of iterations.

The performance accomplished by 60 trees is nearly optimal: even if substantially more trees were trained, only slight improvement on performance could be achieved. For instance, the precision achieved by 120 trees is only 1.5% higher than that attained by 60 trees. And hence, we set the tree number to 60 in all the following experiments.

5.2 Performance of PENDIGITS Queries

We run 3 iterations for each PENDIGITS query, and present the performance of IRF [4] and OPL in Figure 8.

Figure 8 (a) - (d) present precision-recall curves of IRF and OPL when 0-3 iterations are run. Figure 8 (e) - (f) compares the performance of IRF and OPL after 2 iterations are run. In Figure 8 (e) - (f), *Plain* denotes the plain-Euclidean-distance-based database search approach, which does not employ relevance feedback. From the figures, we can draw the following observations:

- IRF outperforms OPL noticeably for PENDIGITS Queries. When recall is 10%, IRF improves the precision over OPL by 12% to 13%. The margin on precision between IRF and OPL keep increasing with the improvement of recall. When recall goes up to 50%, the gain on precision achieved by IRF over OPL is 21% to 28%.

- IRF achieves better performance on larger scopes. For instance, when recall is 50%, IRF improves the precision by 11% if the scope is increased from 80 to 120. This indicates that IRF achieves more gain on performance with larger training sets.

- For queries with large relevant point sets, if recall grows large enough, the performance of OPL

[4] For PENDIGITS data set, IRF and ORF are the same, since the number of training samples is seldom larger than 240.

becomes very close to (or even worse than) that of plain Euclidean distance. For instance, when recall is larger than 40%, the OPL is even slightly outperformed by the plain Euclidean distance if more than 2 iterations are run. This is because OPL learns an elliptic query to cover the relevant points within the scope as perfect as possible. Although the elliptic query may locally outperform the circle query (generated by the plain Euclidean distance), it may not cover multiple (nonlinearly distributed) relevant clusters better than a circle.

5.3 Performance of COREL Queries

To evaluate COREL queries, we set k, the number of nearest neighbors returned to the user, to the value of the scope. Therefore, the recall can not correctly measure the performance of COREL queries on smaller scopes. For instance, the maximum achievable recall on the scope of 20 is about $16.7\%(= 20/120)$, since the average size of all semantic categories is about 120. And hence, the recall for the scopes of 20 and 40 are not presented in this experiment.

For each COREL query, we run 10 search iterations. Figure 9 shows the average precision and recall after each iteration. It indicates that both IRF and ORF consistently outperform OPL. After 10 iterations, IRF improves the precision over OPL by 22.6% to 31%. As to the recall, the gain of IRF over OPL is 20.2% to 25.1% increase. This indicates that our method achieves remarkable improvement over OPL by employing random forests in database search.

Experimental results presented so far demonstrate that our method achieves noticeable improvement on performance over OPL. The advantage of our method is that it can learn the query concept from training samples and find multiple nonlinear clusters of relevant points. This is definitely superior to the traditional relevance feedback approaches like OPL, which only performs optimally when relevant points form a single normal-distributed cluster in the data space.

For scopes of 20 and 40, the IRF and ORF are the same, since the number of the training samples seldom gets larger than N_0 (= 240). But for scopes of 80 and 120, they are quite different: For sample sets whose size is larger than N_0, IRF employs our $BRSR$ resampling method to generate multiple reduced sample sets, and train a subset of the tree classifiers from each reduced set; On the contrary, ORF will always train all the trees from the whole sample set.

Figure 9 (c) - (f) show that ORF outperforms IRF marginally by about 3%. This indicates that overfitting does occur slightly more often with IRF, although it may compensate the overfitting problem by reducing the correlation of generalization errors among the member tree classifiers (see Section 3.3). However, IRF is the most cost-effective method among all the three tested approaches, since it runs much more efficiently than ORF and OPL on large-sized training sets (see Section 5.4).

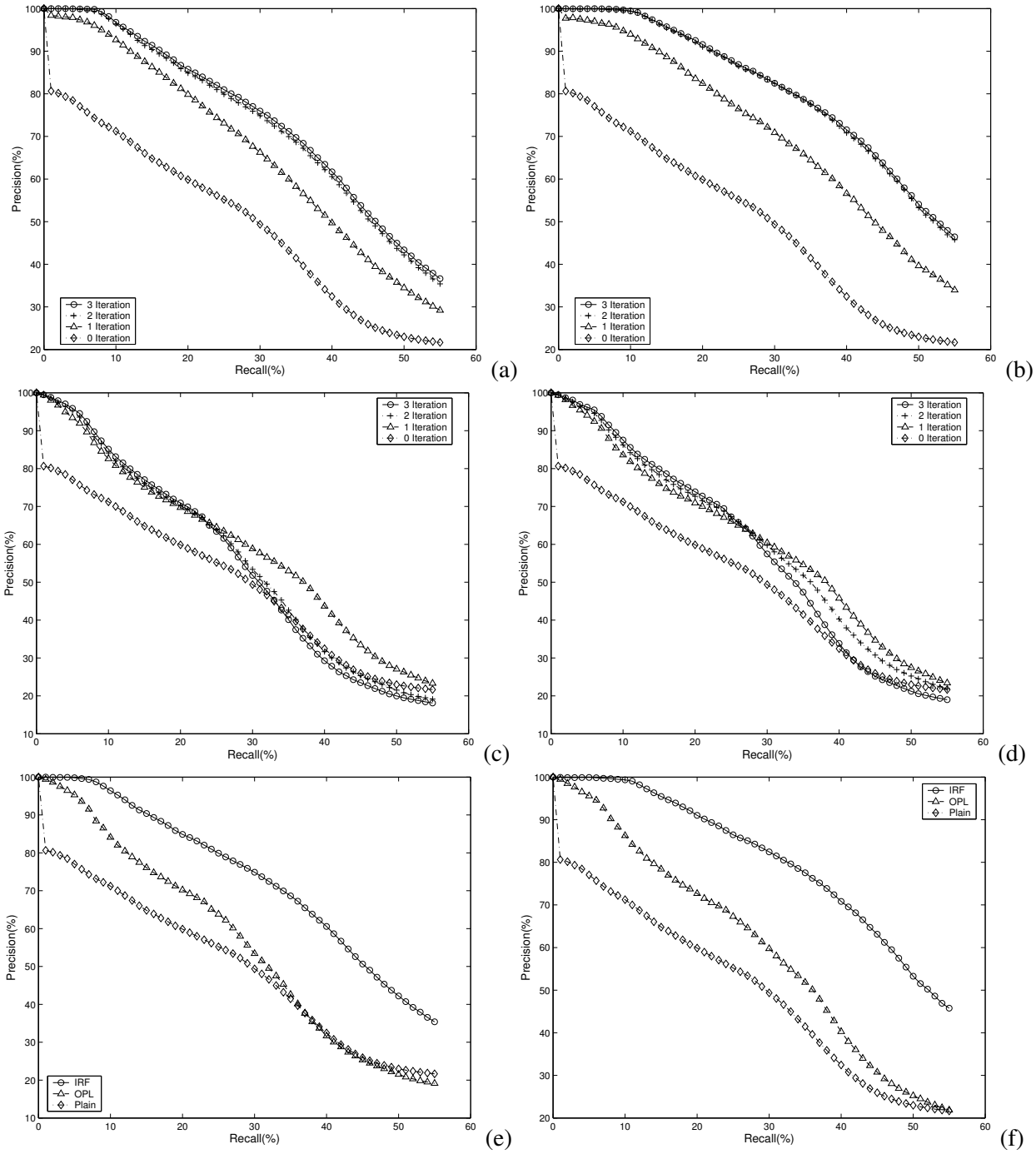


Figure 8: A comparison of OPL and our method using PENDIGITS queries. (a) and (b) present the precision-recall curves of IRF for the scopes of 80 and 120, respectively. (c) and (d) demonstrate the precision-recall curves of OPL for the scopes of 80 and 120, respectively. (e) and (f) compare the precision-recall curves of IRF and OPL for the scopes of 80 and 120, respectively.

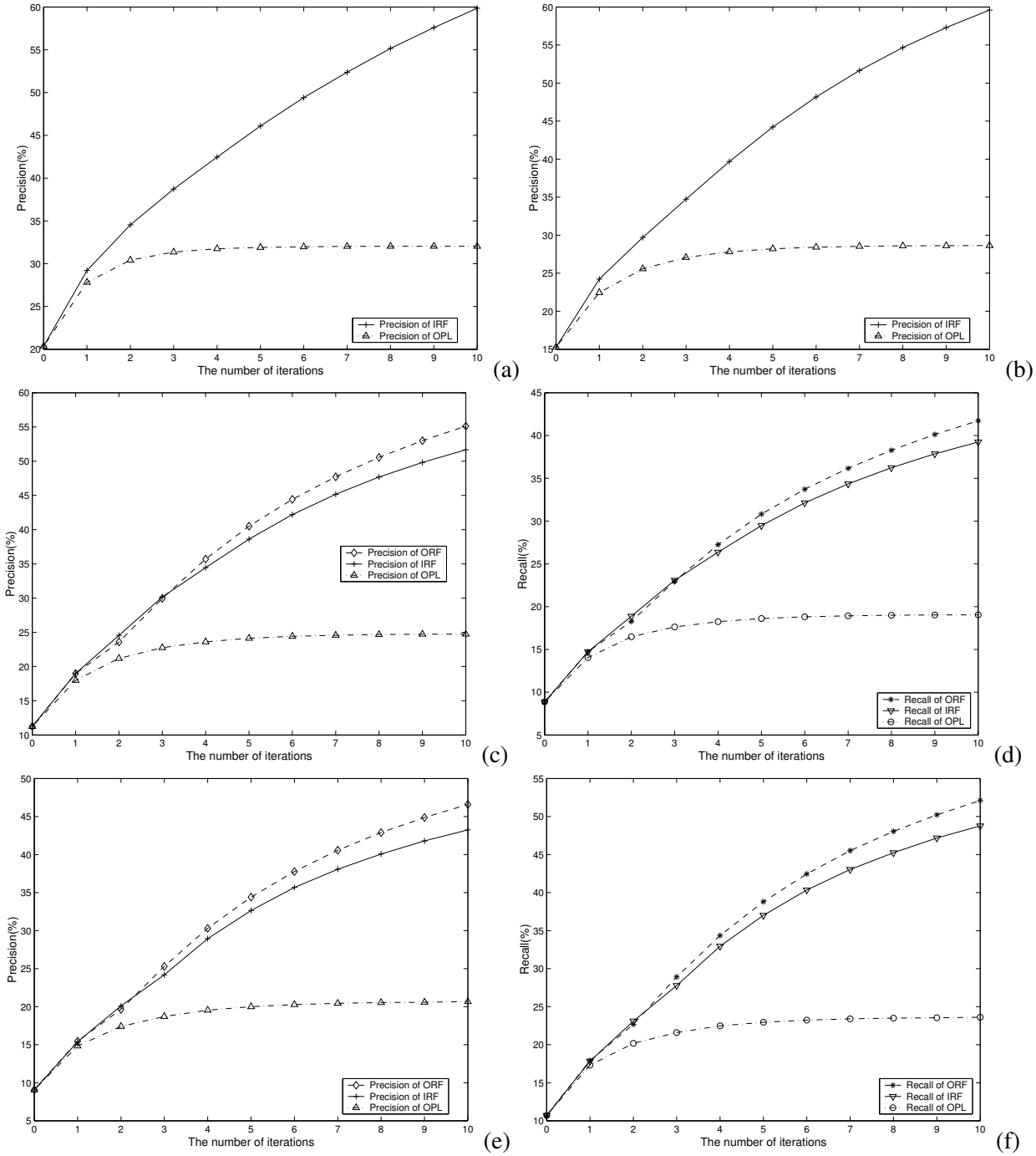
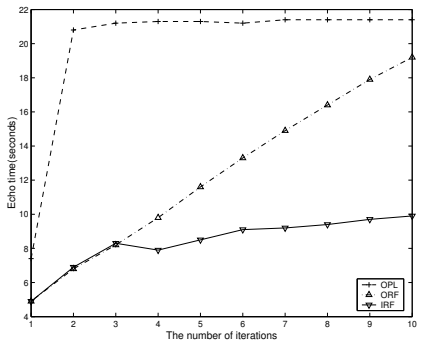
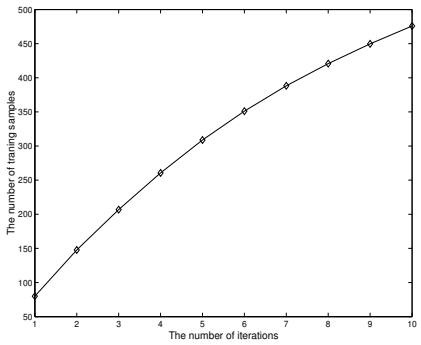


Figure 9: A comparison of OPL and our method using COREL queries. (a) and (b) illustrate the precisions for scopes of 20, 40. (c) and (d) present the precision and recall for the scope of 80. (e) and (f) demonstrate the precision and recall for the scope of 120.

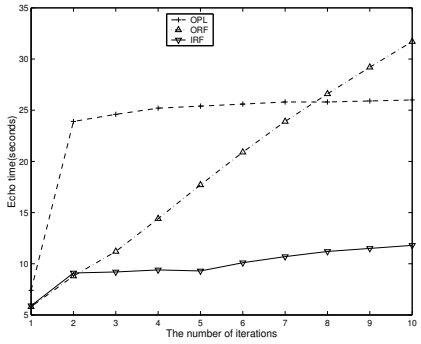
5.4 Efficiency



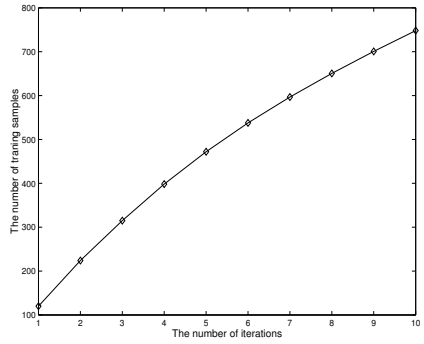
(a) Echo time of scope 80;



(b) Sample number of scope 80.



(a) Echo time of scope 120;



(b) Sample number of scope 120.

Figure 10: Efficiency of different methods.

Due to the lower dimensions of the PENDIGITS data set, all three tested methods run very efficiently on PENDIGITS queries. So, we compare the efficiency of OPL, ORF and IRF with CORE queries. All the experiments are run on a SUN Ultra 80 with a 450MHZ CPU and 1GB memory.

In high-dimensional data spaces, the OPL [16, 21] approach is quite expensive, since its computational cost increases cubically with the dimensions of the data space. With the two-level structure proposed by Rui *et al.* [21], its cost reduces to only cubically related to the maximum dimensions of all image features used. For instance, the computational cost of OPL is cubically related to 64 in our experiments, since 64 is the maximum dimensions of all the image features we used. However, the reduced computational cost is still rather expensive, since the dimensions of each image feature can be as high as hundreds.

Figure 10 compares the efficiency of OPL, ORF and IRF for scopes of 80 and 120. Figure 10 (c) and (d) show that the number of training samples increases sub-linearly with the number of iterations. For OPL, the average echo time for the first iteration is about 7 seconds, and it jumps dramatically to more than 20 seconds from the second iteration. As to the the ORF, its echo time increases linearly with the number of

iterations. For the 10th iteration, the echo time of OPL is about 21 seconds for the scope of 80, and 25 seconds for the scope of 120;The echo time of ORF is about 19 seconds for the scope of 80, and 32 seconds for the scope of 120.

By employing our $BRSR$ resampling method, the echo time of the IRF is always less than or around 10 to 11 seconds, and it runs 2 to 3 times more efficient than both OPL and ORF. So, IRF is the most cost-effective approach among all the three compared methods.

6 Conclusions

In this paper, we presented an adaptive pattern discovery method to search high dimensional data spaces. With our method, the user is allowed to participate in the database search by labeling the returned records as relevant or irrelevant. By using user-labeled records as training samples, our method employs an online pattern discovery technique to learn the distribution patterns of relevant points, and drastically reduce irrelevant data points in the data set. From the reduced data set, our approach refines its previous search results by returning the top-k nearest neighbors (of the query) to the user.

To achieve the adaptive pattern discovery, we employ a pattern classification algorithm called random forests, which is a machine learning algorithm with proven good performance on many traditional classification problems. The random forest is a collection of tree classifiers trained with the given train samples, and it lets all the member classifiers vote for the most popular class when classifying input data points.

To generalize the original random forests algorithm for adaptive pattern discovery, we present a novel two-level resampling method, called $BRSR$, which generates multiple reduced training sets from large-sized sample sets, and trains a subset of trees from each reduced set.

Extensive experimental results on real-world databases demonstrate that our method achieves dramatic improvement on performance over OPL – one of the previously well-known relevance feedback methods. Moreover, with appropriate chosen N_0 – the threshold for the size of the each reduced sample set, our interactive random forest achieves noticeable gains in efficiency over the original random forests.

References

[1] D.A. Keim A. Hinneburg, C.C. Aggarwal. What is the nearest neighbor in high dimensional spaces? In *VLDB Conference*, 2000.

[2] Charu C. Aggarwal. Towards meaningful high-dimensional nearest neighbor search by human-computer interaction. In *Proc. of the 18th Int. Conf. on Data Engineering*, Feb. 2002.

[3] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of ACM-SIGMOD*, May 1990.

[4] S. Berchtold, D. A. Keim, and H.-P.Kriegel. The x-tree: An index structure for high-dimensional data. In *Proc. of the 22th VLDB Conference*, 1996.

[5] K. Beyer, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? In *ICDT Conference*, 1999.

[6] S. Brandt, J. Laaksonen, and E. Oja. Statistical shape features in content-based image retrieval. In *Proceeding of Internationl Conference on Pattern Recognition 2000*, Sept. 2000.

[7] L. Breiman. Bagging predictors. *Machine Learning*, 24:123 – 140, 1997.

[8] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984.

[9] Leo Breiman. Random forests–random features. Technical Report 567, Department of Statistics, University of California,Berkeley, September 1999.

[10] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proc. of the 26th VLDB Conference*, 2000.

[11] Chung-Min Chen and Yibei Ling. A sampling-based estimator for top-k selection query. In *Proc. of the 18th Int. Conf. on Data Engineering*, Feb. 2002.

[12] B. Enfron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, 1993.

[13] M. Annamalai et al. Indexing images in oracle 8i. In *Proc. of SIGMOD Conf.,2000*, 2000.

[14] Y. Freund and R.E. Shapire. A decision-theoretic generalization of online learning and an application to boosting. *J. of Comp. and Sys. Sci.*, 55(1), 1997.

[15] A. Hineburg, C.C Aggarwal, and D.A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proc. of the 26th VLDB Conference*, 2000.

[16] Y. Ishikawa, R. Subramanya, and C. Faloutsos. Mindreader: Query databases through multiple examples. In *Proc. of the 24th VLDB Conference*, 1998.

[17] Norio Katayama and Shinich Statoh. Distintiveness-sensitive nearest-neighbor search for efficient similarity retrieval of multimedia information. In *Proc. of the 17th Int. Conf. on Data Engineering*, April 2001.

[18] W. Y. Ma and H. Zhang. *Handbook of Multimedia Computing*, chapter Content-Based Image Indexing and Retrieval, pages 19–20. London: Routledge, 1998.

[19] Greg Pass, Ramin Zabih, and Justin Miller. Comparing images using color coherence vectors. In *Proceedings of ACM Multimedia 96*, pages 65–73, Boston MA USA, 1996.

[20] K. Porkaew, S. Mehrotra, and M. Ortega. Query reformulation for multimedia retrieval in mars. In *ICMCS*, pages 747 – 751, 1999.

[21] Y. Rui and T.S. Huang. Optimizing learning in image retrieval. *IEEE Conf. on CVPR*, June 2000.

[22] Y. Rui, T.S. Huang, and S. Mehrotra. Content-based image retrieval with relevance feedback in mars. In *Proc. IEEE Int. Conf. on Image Proc.*, 1997.

[23] Z. Su, S. Li, and H. Zhang. Extraction of feature subspaces for content-based retrieval using relevance feedback. In *Proc. of ACM Multimedia*, 2001.

[24] T. Wang, Y. Rui, and S.M. Hu. Optimizing adaptive learning for image retrieval. *IEEE Conference on CVPR*, 1:1140 – 1147, June 2001.

[25] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity search methods in high-dimensional spaces. In *Proc. of the 24th VLDB Conference*, 1998.

[26] L. Wu, C. Faloutsos, K. Sycara, and T.R. Payne. Falcon: Feedback adaptive loop for content-based retrieval. In *Proc. of the 26th VLDB Conference*, September 2000.

[27] Yimin Wu and Aidong Zhang. A feature re-weighting approach for relevance feedback in image retrieval. In *Proc. IEEE Int. Conf. on Image Proc.*, 2002.