Incremental Query Evaluation for Support Vector Machines

Danzhou Liu  Kien A. Hua
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, Florida 32816, USA
{dzliu, kienhua}@cs.ucf.edu

ABSTRACT
Support vector machines (SVMs) have been widely used in multimedia retrieval to learn a concept in order to find the best matches. In such a SVM active learning environment, the system first processes \( k \) sampling queries and \( \text{top-} k \) uncertain queries to select the candidate data items for training. The user’s \( \text{top-} k \) relevant queries are then evaluated to compute the answer. This approach has shown to be effective. However, it suffers from the scalability problem associated with larger database sizes. To address this limitation, we propose an incremental query evaluation technique for these three types of queries. Based on the observation that most queries are not revised dramatically during the iterative evaluation, the proposed technique reuses the results of previous queries to reduce the computation cost. Furthermore, this technique takes advantage of a tuned index structure to efficiently prune irrelevant data. As a result, only a small portion of the dataset needs to be accessed for query processing. This index structure also provides an inexpensive way to process the set of candidates to evaluate the final query result. This technique can work with different kernel functions and kernel parameters. Our experimental results indicate that the proposed technique significantly reduces the overall computation cost, and offers a promising solution to the scalability issue.

Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—Query Formulation, Retrieval Models, Search Process, Selection Process.

General Terms

Keywords
Support Vector Machines, Active Learning, Multimedia Retrieval, Relevance Feedback.

1. INTRODUCTION
Support Vector Machines (SVMs) have been widely used in various applications. In particularly, to address the semantic gap and the user’s subjectivity in multimedia retrieval, relevance feedback coupled with SVMs is typically used to learn a classifier for each user’s query. For example, in content-based image retrieval (CBIR) systems, low-level visual image features (e.g., color, texture, and shape) are automatically extracted for image descriptions and indexing purposes. To search for desirable images, a user can mark returned images as positive or negative, which are then fed back into the system to train a SVM classifier. The system returns a set of images based on its best estimate for further feedback. This process is repeated until the user is satisfied with the query result. Such systems have been shown to be effective for many practical CBIR applications.

In a SVM active learning environment, the following three typical queries are involved: \( k \) sampling query, \( \text{top-} k \) uncertain query, and \( \text{top-} k \) relevant query. Specifically, \( k \) sampling query is to randomly retrieve \( k \) testing instances for user’s feedback; and it is desired that these sampled instances capture the data distribution well (i.e., they are good representatives). \( \text{Top-} k \) uncertain query is to retrieve the \( k \) testing instances closest to the hyperplane. Since these instances are considered most uncertain and informative, they are strongly recommended for the next round of feedback. \( \text{Top-} k \) relevant query is to retrieve the \( k \) farthest instances from the hyperplane in the relevant half-plane. These instances are the most relevant instances based on the learned SVM classifier. Since each feedback iteration typically changes both the transformed space and the separating hyperplane, traditional query evaluation techniques cannot be straightforwardly applied. As a consequence, most existing CBIR systems with SVMs have to linearly scan the entire image set to evaluate both \( \text{top-} k \) uncertain and \( \text{top-} k \) relevant queries, resulting in the scalability issue for large collections of multimedia data. To address this scalability issue, Panda and Chang [5] mapped \( \text{top-} k \) uncertain and \( \text{top-} k \) relevant queries into range queries in the original space, allowing for the reuse of existing index structures. Our query processing technique (see Section 2) is inspired by their work, and is more efficient by reusing the results of previous queries instead of query evaluation from scratch, optimizing the underlying index structures, estimating the bounding box without expensive SVM clustering, and proposing faster query evaluation for range queries. Experimental results in Section 3 show that our approach achieves significant savings in terms of disk I/O costs and
execution time compared to the above approach.

2. THE PROPOSED INCREMENTAL QUERY EVALUATION TECHNIQUE

In this section, we discuss the proposed query evaluation technique. Specifically, we first present index construction and tuning to facilitate efficient query evaluation in Section 2.1. Then, we discuss in details our incremental query evaluation technique in Section 2.2.

2.1 Index Construction and Tuning

Our index structure is constructed and tuned as follows:

Hierarchical Clustering: A hierarchical clustering technique, similar to the R*-tree [1], is used to organize the entire image database into a hierarchical tree structure.

Information Augmenting: We traverse the tree in a postorder fashion. In an original R*-tree, an internal node contains an array of node entries. Each node entry is a pair (mbb, node-id), where mbb is the minimum bounding box (MBB) that spatially contains the MBBs in the child node, with node-id as the child node address. In our index structure, each node entry is extended to be a tuple (mbb, node-id, imageID-range), where imageID-range refers to the range of image identifications contained in the pointed child node and imageID-range ⊆ [1, |S|] where |S| is the cardinality of the whole image database.

Index Tuning: The design of most existing hierarchical index structures (e.g., R*-tree) usually overlooks the differences between sequential and random accesses. Since the disk pages allocated to sibling nodes are often not physically consecutive (typically a disk page contains only one node), a query may incur a large number of random accesses even for each feedback iteration. To reduce the number of disk random accesses, we use the Hilbert curve [6] for disk page allocation. Specifically, we create a tuned index structure as follows: we traverse the non-tuned index structure in a breadth-first fashion, and then create a tuned index structure with the disk page allocation almost following the traversal order except for the children nodes in the same node. For the children nodes in the same node, we allocate them to the disk in the order of the Hilbert curve values of their centers.

2.2 Efficient Query Evaluation

We discuss our query processing technique EVALUATEQUERY(Q) on top of the above index structure for the three types of
queries (i.e., $k$ sampling queries, top-$k$ uncertain query, and top-$k$ relevant queries) used in SVM active learning. In this paper, we focus on the two-class SVM active learning. Given a data set $X$ that consists of vectors in a metric space $M$. Among $X$, the training data set is denoted as $X_t = \{x_{t,1}, \ldots, x_{t,n}\}$ with the corresponding labels $Y_t = \{y_{1}, \ldots, y_{n}\}$, where $y_i \in \{-1, 1\}$. The testing data set is denoted as $X_u = \{x_{u,1}, \ldots, x_{u,m}\}$, and $X = X \setminus X_t$. During the query-concept learning phase, SVM typically transfers $X_t$ from $M$ into a feature space $F$, and derives a hyperplane separating the relevant training instances (i.e., those with the label 1) from irrelevant ones (i.e., those with the label -1), and achieving the largest margin [2]. The weights $W = \{\alpha_1, \ldots, \alpha_n\}$ associated with $X_t$ are determined accordingly. Those testing instances with $\alpha_i > 0$ are called support vectors, and they are in fact the closest points to the hyperplane. The class membership of a testing instance $x_{u,j}$ can be predicted by the following function: $S(x_{u,j}) = \sum_{i=1}^{n} \alpha_i y_i K(x_{t,i}, x_{u,j}) + b$, where $K$ is a kernel function. If $S(x_{u,j}) \geq 0$, $x_{u,j}$ is classified as +1; otherwise, -1. In fact, a top-1 relevant query is to retrieve $k$ instances with the largest values of $S$ in $X$, and a top-$k$ uncertain query is to retrieve the $k$ instances with the smallest absolute values of $S$ in $X_u$. Note that the query cost is the sum of disk seek (including cylinder seek and rotation), data transfer and CPU time, in which seek time dominates the total query cost. Figure 2 lays out our query processing technique, designed to minimize the disk I/O cost.

For $k$ sampling queries, we just need to retrieve the root node (in line 1) that contains all possible image IDs. If $k$ is relatively small, we sample instances from $k$ different leaf nodes in order to make the sampled instances as representative as possible (in line 6). Otherwise, random sampling can be performed to retrieve the query result (in line 8).

For top-$k$ uncertain queries, we first determine the positive and negative support vectors in $X_u$ (in line 12). Specifically, the positive support vectors are those testing instances with $\alpha_i > 0$ and $y_i = 1$ while the positive support vectors are those with $\alpha_i > 0$ and $y_i = -1$. Top $k$ uncertain query aims to retrieve the $k$ instances in $X_u$ closest to the hyperplane. These instances typically lie between positive support vectors and negative support vectors. Therefore, a bounding box $B$ that covers both positive and negative support vectors has a high probability to cover the desired query result. After deriving $B$, we can partition $B$ into multiple ranges (in line 14) to eliminate the empty space by some partitioning strategies, such as Equi-Count, Equi-Area, Min-Skew and Min-Overlap. The Equi-Count partitioning strategy creates ranges containing roughly the same number of instances. The Equi-Area partitioning strategy creates ranges having the same area. The Min-Skew partitioning strategy divides $B$ into ranges such that each range contains uniformly distributed instances. The Min-Overlap partitioning strategy creates ranges that have minimal overlaps among them. We can adopt any of the above partitioning strategies. In our experiments, we have implemented the Equi-Area partitioning strategy. After a set of ranges is determined, we can avoid some unnecessary range queries by eliminating those ranges within the ranges of previous queries (in line 15). Such incremental strategy works because of the observation that most queries are not revised dramatically during the iterative evaluation. Of course, this strategy incurs memory overhead because we need to buffer the previous results in memory. Considering the performance gain shown in Section 4, this overhead is still acceptable. Then, $\text{EVALUATE}$ is called to perform multiple range queries to get the results $S$ (in line 16). Finally, we obtain the query result $S_k$ by calculating the values of $S(x_{u,j}) = \sum_{i=1}^{n} \alpha_i y_i K(x_{t,i}, x_{u,j}) + b$ for $\forall x_{u,j} \in (S \cup S_{prev}) \cap X_u$, and sorting these values to get $k$ instances with the smallest absolute values (in line 17).

To evaluate top-$k$ relevant queries, we first determine the relevant instances (i.e., those training instances with $y_i = 1$) (in line 19). Then we derive the minimum bounding box $B$ for these relevant instances. Because top-$k$ relevant query is to retrieve $k$ farthest instances to the hyperplane in the relevant half-plane, $B$ is needed to expand to cover these instances in some cases. One way is to expand $B$ at a given ratio, and another way is to use sampling instances to estimate the desired range. After expanding $B$, we can partition $B$ into multiple ranges, and achieve incremental query evaluation by removing some ranges within the ranges of previous queries (in line 23) similarly as for top-$k$ uncertain queries. Then, we call $\text{EVALUATE}$ to perform multiple range queries. Finally, query result $S_k$ is determined by calculating the values of $S(x_{u,j}) = \sum_{i=1}^{n} \alpha_i y_i K(x_{t,i}, x_{u,j}) + b$ for $\forall x_{t,j} \in (S \cup S_{prev})$, and sorting these values to obtain $k$ instances with the largest values (in line 25).

In addition, the algorithm for multiple range queries (i.e., $\text{EVALUATE}$, see Figure 3) has some important features. Firstly, this algorithm accesses each node at most once for multiple ranges by performing the breadth-first search to traverse the proposed index structure. Secondly, such breadth-first search can take advantage of our index structure to achieve sequential access during the search process, thus reducing the disk seek time significantly.

**Figure 3: Algorithm for multiple range queries**
3. EXPERIMENTS

In this section, we evaluate the effectiveness of the proposed query processing technique described in Section 3. Our dataset consists of more than 68,040 images from the COREL library. There are a total of 37 visual image features. The COREL images have been classified into distinct categories by domain professionals, and each category contains about 100 images. For each chosen category, 50% of images were used as the training data. We chose LIBSVM [3] with the Gaussian kernel for SVM learning. The node size of the original R*-tree and our index structure were both set to 4KB, and both had three levels in our experimental settings. We compare the performance of the proposed query processing technique on top of our index structure (denoted as TNEW) against the existing technique with R*-tree (denoted as TOLD). Specifically, our methods to evaluate k sampling queries, top-k uncertain queries, top-k relevant queries are from line 4 to 10, 11 to 17, and 18 to 25 in Figure 2, respectively. The existing counterparts are proposed in [4], [5], and [5], respectively. The experiments were performed on a 3.4-GHz Pentium IV-based computer with 1.5GB of RAM, and the results are averaged over 100 runs.

Figures 4 to 6 show that TNEW significantly outperforms TOLD for answering three types of queries in terms of disk accesses with different k ∈ {5, 15, 20, 25, 35, 50}, and the performance gap widens as k increases. Clearly, the total number of disk access increases as k increases. As shown in Figure 4, TOLD performs about five times more disk accesses than TNEW when k = 5, 58 times when k = 25, and 120 times when k = 50 for k sampling queries. This figure shows that TNEW is independent of the number of sample points (i.e., k) because TNEW just needs to access the root node of our index structure, resulting in only one disk access for answering a sampling query. On the other hand, TOLD is proportional to k, which is because TOLD has to traverse the R*-tree to obtain sample points one by one, incurring almost 3 disk accesses per sample point. If a sampling point is not met the criteria, another traversal is needed. For top-k uncertain queries, TOLD performs about twice disk accesses compared to TNEW when k = 5, and about three times when k = 50 (see Figure 5). For top k relevant queries, TNEW performs up to about three times better (see Figure 6). More importantly, the curves of TNEW are quite low and flat, indicating that it can support a large k with much less disk I/O overhead. The performance difference between TNEW and TOLD confirms that the proposed query processing technique reduces the disk I/O cost and execution time significantly by incremental query evaluation, taking advantage of our index structure, and evaluating efficiently those three types of queries.

4. CONCLUSIONS

Although support vector machines have been shown to be effective for multimedia retrieval, it suffers from the scalability problems associated with larger database sizes. This important limitation is addressed in this paper by proposing a highly efficient query evaluation technique for SVMs. Taking advantage of an index structure tuned for better data clustering, the proposed technique answers k sampling queries on the fly, transforms top-k uncertain queries and top-k relevant queries into range queries in the original space, and then evaluates these range queries efficiently. More importantly, by reusing the results of previous queries, the proposed technique can save query evaluation cost much more. This approach is not affected by the changes of kernels and kernel parameters of SVMs. The experimental results indicate that our approach significantly reduces the computation time and the number of disk accesses for query evaluation.

5. REFERENCES


Figure 4: Disk accesses of k sampling queries Figure 5: Disk accesses of top-k uncertain queries Figure 6: Disk accesses of top-k relevant queries