Support Vector Machine Active Learning for Image Retrieval

Simon Tong
Department of Computer Science
Stanford University
Sanford, CA 94305
simon.tong@cs.stanford.edu

Edward Chang
Electrical & Computer Engineering
University of California
Santa Barbara, CA 93106
echang@ece.ucsb.edu

ABSTRACT

Relevance feedback is often a critical component when designing image databases. With these databases it is difficult to specify queries directly and explicitly. Relevance feedback interactively determinines a user's desired output or *query concept* by asking the user whether certain proposed images are relevant or not. For a relevance feedback algorithm to be effective, it must grasp a user's query concept accurately and quickly, while also only asking the user to label a small number of images. We propose the use of a *support vector machine active learning* algorithm for conducting effective relevance feedback for image retrieval. The algorithm selects the most informative images to query a user and quickly learns a boundary that separates the images that satisfy the user's query concept from the rest of the dataset. Experimental results show that our algorithm achieves significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback.

Keywords

active learning, image retrieval, query concept, relevance feedback, support vector machines.

1. INTRODUCTION

One key design task, when constructing image databases, is the creation of an effective relevance feedback component. While it is sometimes possible to arrange images within an image database by creating a hierarchy, or by hand-labeling each image with descriptive words, it is often time-consuming, costly and subjective. Alternatively, requiring the end-user to specify an image query in terms of low level features (such as color and spatial relationships) is challenging to the end-user, because an image query is hard to articulate, and articulation can again be subjective.

Thus, there is a need for a way to allow a user to implicitly inform a database of his or her desired output or *query concept*. To address this requirement, relevance feedback can be used as a query refinement scheme to derive or learn a user's query concept. To solicit feedback, the refinement scheme displays a few image instances and the user labels each image as "relevant" or "not relevant." Based on the answers, another set of images from the database are brought up to the user for labeling. After some number of such querying rounds, the refinement scheme returns a number of items in the database that it believes will be of interest to the user.

The construction of such a query refinement scheme (we call it a query concept learner or *learner* hereafter) can be regarded as a machine learning task. In particular, it can be seen as a case of *pool-based active learning* [19, 23]. In pool-based active learning the learner has access to a pool of unlabeled data and can request the user's label for a certain number of instances in the pool. In the image retrieval domain, the unlabeled pool would be the entire database of images. An instance would be an image, and the two possible labelings of an image would be "relevant" and "not relevant." The goal for the learner is to learn the user's *query concept* — in other words the goal is to give a label to each image within the database such that for any image, the learner's labeling and the user's labeling will agree.

The main issue with active learning is finding a way to choose informative images within the pool to ask the user to label. We call such a request for the label of an image a *pool-query*. Most machine learning algorithms are *passive* in the sense that they are generally applied using a randomly selected training set. The key idea with *active* learning is that it should choose its next pool-query based upon the past answers to previous pool-queries.

In general, and for the image retrieval task in particular, such a learner must meet two critical design goals. First, the learner must learn target concepts accurately. Second, the learner must grasp a concept quickly, with only a small number of labeled instances, since most users do not wait around to provide a great deal of feedback. In this study, we propose using a *support vector machine active learner* (SVM_{Active}) to achieve these goals. SVM_{Active} combines active learning with support vector machines (SVMs). SVMs [35, 1] have met with significant success in numerous real-world learning tasks. Like most machine learning algorithms, they are generally applied using a randomly selected training set which is not very useful in the relevance feedback setting. Recently, general purpose methods for active learning with SVMs have been independently developed by a number of researchers [2, 29, 34]. We shall use the work and theoretical motivation of [34] on active learning with SVMs to extend the use of support vector machines to the task of relevance feedback for image databases.

Intuitively, SVM_{Active} works by combining the following three ideas:

1. SVM_{Active} regards the task of learning a target concept as one of learning a SVM binary classifier. An SVM captures the query concept by separating the relevant images from the irrelevant images with a hyperplane in a projected space, usually a very high-dimensional one. The projected points on one side of the hyperplane are considered relevant to the query concept and the rest irrelevant.
2. SVM_{Active} learns the classifier **quickly** via active learning. The active part of SVM_{Active} selects the most informative instances with which to train the SVM classifier. This step ensures fast convergence to the query concept in a small number of feedback rounds.
3. Once the classifier is trained, SVM_{Active} returns the top-k most relevant images. These are the k images farthest from the hyperplane on the query concept side.

In summary, our contributions are as follows:

- The use of SVM active learning for image retrieval. We show that combining SVMs with an active learning component can produce a learner that is particularly well suited to the query refinement task in image retrieval, significantly outperforming traditional methods.
- The multi-resolution image feature organization. We show that organizing image features in different resolutions gives the learner the flexibility to model subjective perception and to satisfy a variety of search tasks. Using this representation for images, our learner can quickly converge to target query concepts.

The rest of this paper is organized into seven sections. Section 2 introduces SVMs. Section 3 then introduces the notion of a *version space* which in Section 4 provides theoretical motivation for a method for performing active learning with SVMs. Section 5 depicts our multi-resolution image characterization. In Section 6, we report experimental results showing that our SVM active learner significantly outperforms traditional methods. Section 7 surveys related work. Finally, we offer our conclusions in Section 8.

2. SUPPORT VECTOR MACHINES

Support vector machines are a core machine learning technology. They have strong theoretical foundations and excellent empirical successes. They have been applied to tasks such as handwritten digit recognition [36], object recognition [26], and text classification [15].

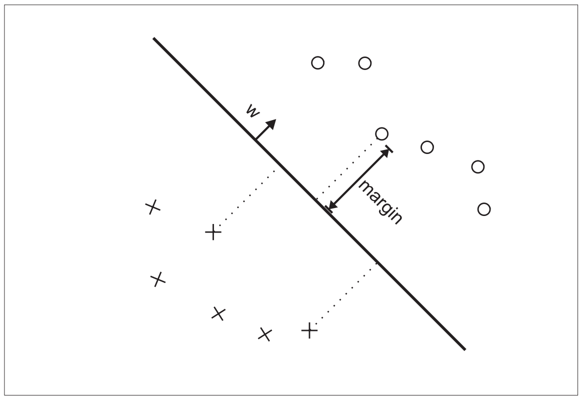


Figure 1: A simple linear Support Vector Machine

We shall consider SVMs in the binary classification setting. We are given training data $\{\mathbf{x}_1 \dots \mathbf{x}_n\}$ that are vectors in some space $\mathcal{X} \subseteq \mathbb{R}^d$. We are also given their labels $\{y_1 \dots y_n\}$ where $y_i \in \{-1, 1\}$. In their simplest form, SVMs are hyperplanes that separate the training data by a maximal margin (see Fig. 1). All vectors lying on one side of the hyperplane are labeled as -1, and all vectors lying on the other side are labeled as 1. The training instances that lie closest to the hyperplane are called *support vectors*. More generally, SVMs allow one to project the original training data in space $\mathcal{X}$ to a higher dimensional feature space $\mathcal{F}$ via a Mercer kernel operator K. In other words, we consider the set of classifiers of the form: $f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x})$. When $f(\mathbf{x}) \geq 0$ we classify $\mathbf{x}$ as $+1$, otherwise we classify $\mathbf{x}$ as -1.

When K satisfies Mercer's condition [1] we can write: $K(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$ where $\Phi : \mathcal{X} \rightarrow \mathcal{F}$ and "$\cdot$" denotes an inner product. We can then rewrite f as:

$$f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}), \text{ where } \mathbf{w} = \sum_{i=1}^{n} \alpha_i \Phi(\mathbf{x}_i). \qquad (1)$$

Thus, by using K we are implicitly projecting the training data into a different (often higher dimensional) feature space $\mathcal{F}$. The SVM then computes the α_is that correspond to the maximal margin hyperplane in $\mathcal{F}$. By choosing different kernel functions we can implicitly project the training data from $\mathcal{X}$ into spaces $\mathcal{F}$ for which hyperplanes in $\mathcal{F}$ correspond to more complex decision boundaries in the original space $\mathcal{X}$.

Two commonly used kernels are the polynomial kernel $K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^p$ which induces polynomial boundaries of degree p in the original space $\mathcal{X}$ and the radial basis function kernel $K(\mathbf{u}, \mathbf{v}) = (e^{-\gamma(\mathbf{u}-\mathbf{v})\cdot(\mathbf{u}-\mathbf{v})})$ which induces boundaries by placing weighted Gaussians upon key training instances. In the remainder of this paper we will assume that the modulus of the training data feature vectors are constant, i.e., for all training instances $\mathbf{x}_i$, $\|\Phi(\mathbf{x}_i)\| = \lambda$ for some fixed λ. The quantity $\|\Phi(\mathbf{x}_i)\|$ is always constant for radial basis function kernels, and so the assumption has no effect for this kernel. For $\|\Phi(\mathbf{x}_i)\|$ to be constant with the polynomial kernels we require that $\|\mathbf{x}_i\|$ be constant. It is possible to relax this constraint on $\Phi(\mathbf{x}_i)$. We shall discuss this option at the end of Section 4.

3. VERSION SPACE

Given a set of labeled training data and a Mercer kernel K, there is a set of hyperplanes that separate the data in the induced feature space $\mathcal{F}$. We call this set of consistent hyperplanes or *hypotheses* the *version space* [24]. In other words, hypothesis f is in version space if for every training instance $\mathbf{x}_i$ with label y_i we have that $f(\mathbf{x}_i) > 0$ if $y_i = 1$ and $f(\mathbf{x}_i) < 0$ if $y_i = -1$. More formally:

Definition 3.1. *Our set of possible hypotheses is given as:*

$$\mathcal{H} = \left\{ f \mid f(\mathbf{x}) = \frac{\mathbf{w} \cdot \Phi(\mathbf{x})}{\|\mathbf{w}\|} \text{ where } \mathbf{w} \in \mathcal{W} \right\},$$

where our parameter space $\mathcal{W}$ *is simply equal to* $\mathcal{F}$. *The* Version space, $\mathcal{V}$ *is then defined as:*

$$\mathcal{V} = \{ f \in \mathcal{H} \mid \forall i \in \{1 \dots n\} \quad y_i f(\mathbf{x}_i) > 0\}.$$

Notice that since $\mathcal{H}$ *is a set of hyperplanes, there is a bijection (an exact correspondence) between unit vectors* $\mathbf{w}$ *and hypotheses* f *in* $\mathcal{H}$. *Thus we will redefine* $\mathcal{V}$ *as:*

$$\mathcal{V} = \{ \mathbf{w} \in \mathcal{W} \mid \|\mathbf{w}\| = 1, \; y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i)) > 0, i = 1 \dots n\}.$$

Note that a version space only exists if the *training* data are linearly separable in the feature space. Thus, we require linear separability of the training data in the feature space. This restriction is much less harsh than it might at first seem. First, the feature space often has a very high dimension and so in many cases it results in the data set being linearly separable. Second, as noted by [31], it is possible to modify any kernel so that the data in the new induced feature space is linearly separable. This is done by redefining for all training instances $\mathbf{x}_i$: $K(\mathbf{x}_i, \mathbf{x}_i) \leftarrow K(\mathbf{x}_i, \mathbf{x}_i) + \nu$ where ν is a positive regularization constant. The effect of this modification is to permit linear non-separability of the training data in the original feature space.

There exists a duality between the feature space $\mathcal{F}$ and the parameter space $\mathcal{W}$ [36, 12] which we shall take advantage of in the next section: points in $\mathcal{F}$ correspond to hyperplanes in $\mathcal{W}$ and *vice versa.*

Clearly, by definition, points in $\mathcal{W}$ correspond to hyperplanes in $\mathcal{F}$. The intuition behind the converse is that observing a training instance $\mathbf{x}_i$ in feature space restricts the set of separating hyperplanes to ones that classify $\mathbf{x}_i$ correctly. In fact, we can show that the set of allowable points $\mathbf{w}$ in $\mathcal{W}$ is restricted to lie on one side of a hyperplane in $\mathcal{W}$. More formally, to show that points in $\mathcal{F}$ correspond to hyperplanes in $\mathcal{W}$, suppose we are given a new training instance $\mathbf{x}_i$ with label y_i. Then any separating hyperplane must satisfy $y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i)) > 0$. Now, instead of viewing $\mathbf{w}$ as the normal vector of a hyperplane in $\mathcal{F}$, think of $y_i\Phi(\mathbf{x}_i)$ as being the normal vector of a hyperplane in $\mathcal{W}$. Thus $y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i)) = \mathbf{w} \cdot y_i\Phi(\mathbf{x}_i) > 0$ defines a half-space in $\mathcal{W}$. Furthermore $\mathbf{w} \cdot y_i\Phi(\mathbf{x}_i) = 0$ defines a hyperplane in $\mathcal{W}$ that acts as one of the boundaries to version space $\mathcal{V}$. Notice that version space is a connected region on the surface of a hypersphere in parameter space. See Figure 3(a) for an example.

SVMs find the hyperplane that maximizes the margin in feature space $\mathcal{F}$. One way to pose this is as follows:

$$\begin{array}{ll} \text{maximize}_{\mathbf{w} \in \mathcal{F}} & \min_i\{y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i))\} \\ \text{subject to:} & \|\mathbf{w}\| = 1 \\ & y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i)) > 0 \quad i = 1 \dots n. \end{array}$$

By having the conditions $\|\mathbf{w}\| = 1$ and $y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i)) > 0$ we cause the solution to lie in version space. Now, we can view the above problem as finding the point $\mathbf{w}$ in version space that maximizes the distance $\min_i\{\mathbf{w} \cdot y_i\Phi(\mathbf{x}_i)\}$. From the duality between feature and parameter space, and since $\|\Phi(\mathbf{x}_i)\| = 1$, then each $y_i\Phi(\mathbf{x}_i)$ is a unit normal vector of a hyperplane in parameter space and each of these hyperplanes delimits the version space. Thus we want to find the point in version space that maximizes the minimum distance to any of the delineating hyperplanes. That is, SVMs find the center of the largest radius hypersphere whose center can be placed in version space and whose surface does not intersect with the hyperplanes corresponding to the labeled instances, as in Figure 3(b). It can be easily shown that the hyperplanes touched by the maximal radius hypersphere correspond to the support vectors and that the radius of the hypersphere is the margin of the SVM.

4. ACTIVE LEARNING

In pool-based active learning we have a pool of unlabeled instances. It is assumed that the instances $\mathbf{x}$ are independently and identically distributed according to some underlying distribution $F(\mathbf{x})$ and the labels are distributed according to some conditional distribution $P(y \mid \mathbf{x})$.

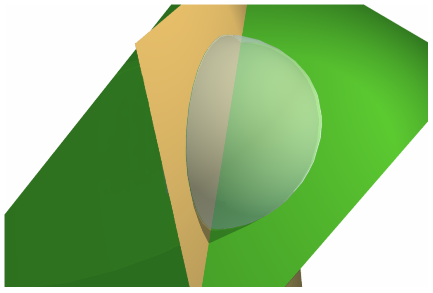
Given an unlabeled pool U, an *active learner* ℓ has three components: (f, q, X). The first component is a classifier, $f : \mathcal{X} \to \{-1, 1\}$, trained on the current set of labeled data X (and possibly unlabeled instances in U too). The second component $q(X)$ is the querying function that, given a current labeled set X, decides which instance in U to query next. The active learner can return a classifier f after each pool-query (*online learning*) or after some fixed number of pool-queries.

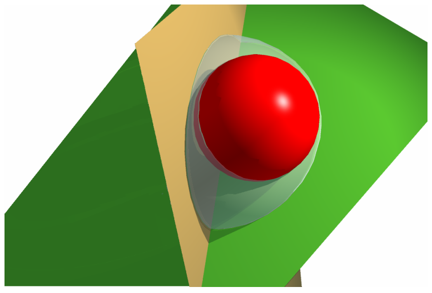
The main difference between an active learner and a regular passive learner is the querying component q. This brings us to the issue of how to choose the next unlabeled instance in the pool to query. We use an approach that queries points so as to attempt to reduce the size of the version space as much as possible. We need one more definition before we can proceed:

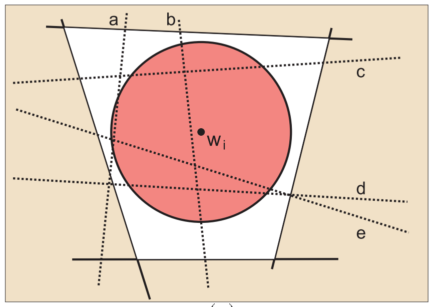
Definition 4.1. *Area*($\mathcal{V}$) *is the surface area that the version space* $\mathcal{V}$ *occupies on the hypersphere* $\|\mathbf{w}\| = 1$.

We wish to reduce the version space as fast as possible. Intuitively, one good way of doing this is to choose a pool-query that halves the version space. More formally, we can use the following lemma to motivate which instances to use as our pool-query:

Lemma 4.2. *(Tong & Koller, 2000) Suppose we have an input space* $\mathcal{X}$, *finite dimensional feature space* $\mathcal{F}$ *(induced*


(a)


(b)



(c)

Figure 2: (a) Version space duality. The surface of the hypersphere represents unit weight vectors. Each of the two hyperplanes corresponds to a labeled training instance. Each hyperplane restricts the area on the hypersphere in which consistent hypotheses can lie. Here version space is the surface segment of the hypersphere closest to the camera. (b) An SVM classifier in version space. The dark embedded sphere is the largest radius sphere whose center lies in version space and whose surface does not intersect with the hyperplanes. The center of the embedded sphere corresponds to the SVM, its radius is the margin of the SVM in $\mathcal{F}$ and the training points corresponding to the hyperplanes that it touches are the support vectors. (c) Simple Margin Method.

via a kernel K), and parameter space $\mathcal{W}$. Suppose active learner ℓ^* always queries instances whose corresponding hyperplanes in parameter space $\mathcal{W}$ halves the area of the current version space. Let ℓ be any other active learner. Denote the version spaces of ℓ^* and ℓ after i pool-queries as $\mathcal{V}_i^*$ and $\mathcal{V}_i$ respectively. Let $\mathcal{P}$ denote the set of all conditional distributions of y given $\mathbf{x}$. Then,

$$\forall i \in \mathbb{N}^+ \quad \sup_{P\in\mathcal{P}} E_P[Area(\mathcal{V}_i^*)] \leq \sup_{P\in\mathcal{P}} E_P[Area(\mathcal{V}_i)],$$

with strict inequality whenever there exists a pool-query $j \in \{1 \ldots i\}$ by ℓ that does not halve version space V_{j-1}.

This lemma says that, for any given number of pool-queries, ℓ^* minimizes the maximum expected size of the version space, where the maximum is taken over all conditional distributions of y given $\mathbf{x}$.

Now, suppose $\mathbf{w}^* \in \mathcal{W}$ is the unit parameter vector corresponding to the SVM that we would have obtained had we known the actual labels of *all* of the data in the pool. We know that $\mathbf{w}^*$ must lie in each of the version spaces $\mathcal{V}_1 \supset \mathcal{V}_2 \supset \mathcal{V}_3 \ldots$, where $\mathcal{V}_i$ denotes the version space after i pool-queries. Thus, by shrinking the size of the version space as much as possible with each pool-query we are reducing as fast as possible the space in which $\mathbf{w}^*$ can lie. Hence, the SVM that we learn from our limited number of pool-queries will lie close to $\mathbf{w}^*$.

This discussion provides motivation for an approach where we query instances that split the current version space into two equal parts as much as possible. Given an unlabeled instance $\mathbf{x}$ from the pool, it is not practical to explicitly compute the sizes of the new version spaces $\mathcal{V}^-$ and $\mathcal{V}^+$ (i.e., the version spaces obtained when $\mathbf{x}$ is labeled as -1 and $+1$ respectively). There is way of approximating this procedure as noted by [34]:

Simple Method. Recall from section 3 that, given data $\{\mathbf{x}_1 \ldots \mathbf{x}_i\}$ and labels $\{y_1 \ldots y_i\}$, the SVM unit vector $\mathbf{w}_i$ obtained from this data is the center of the largest hypersphere that can fit inside the current version space $\mathcal{V}_i$. The position of $\mathbf{w}_i$ in the version space $\mathcal{V}_i$ clearly depends on the shape of the region $\mathcal{V}_i$, however it is often approximately in the center of the version space. Now, we can test each of the unlabeled instances $\mathbf{x}$ in the pool to see how close their corresponding hyperplanes in $\mathcal{W}$ come to the centrally placed $\mathbf{w}_i$. The closer a hyperplane in $\mathcal{W}$ is to the point $\mathbf{w}_i$, the more centrally it is placed in version space, and the more it bisects version space. Thus we can pick the unlabeled instance in the pool whose hyperplane in $\mathcal{W}$ comes closest to the vector $\mathbf{w}_i$. For each unlabeled instance $\mathbf{x}$, the shortest distance between its hyperplane in $\mathcal{W}$ and the vector $\mathbf{w}_i$ is simply the distance between the feature vector $\Phi(\mathbf{x})$ and the hyperplane $\mathbf{w}_i$ in $\mathcal{F}$ — which is easily computed by $|\mathbf{w}_i \cdot \Phi(\mathbf{x})|$. This results in the natural Simple rule:

- Learn an SVM on the existing labeled data and choose as the next instance to query the pool instance that comes closest to the hyperplane in $\mathcal{F}$.

Figure 3(c) presents an illustration. In the stylized picture we have flattened out the surface of the unit weight vector hypersphere that appears in Figure 3(a). The white area is version space $\mathcal{V}_i$ which is bounded by solid lines corresponding to labeled instances. The five dotted lines represent unlabeled instances in the pool. The circle represents the largest radius hypersphere that can fit in version space. Note that the edges of the circle do not touch the solid lines — just as the dark sphere in 3(b) does not meet the hyperplanes on the surface of the larger hypersphere (they meet somewhere under the surface). The instance **b** is closest to the SVM $\mathbf{w}_i$ and so we will choose to query **b**.

As noted by [34] there exist more sophisticated approximations that one can perform. However, these methods are significantly more computationally intensive. For the task of relevance feedback, a fast response time for determining the next image to present to the user is so critically important that these other approximations are not practical.

Our SVM$_{Active}$ image retrieval system uses radial basis function kernels. As noted in Section 2, radial basis function kernels have the property that $\|\Phi(\mathbf{x_i})\| = \lambda$. The Simple querying method can still be used with other kernels when the training data feature vectors do not have a constant modulus, but the motivating explanation no longer holds since the SVM can no longer be viewed as the center of the largest allowable sphere. However, alternative motivations have recently been proposed by Campbell, Cristianini and Smola (2000) that do not require a constraint on the modulus.

For the image retrieval domain, we also have a need for performing multiple pool-queries at the same time. It is not practical to present one image at a time for the user to label — the user is likely to quickly lose patience after a few rounds of pool-querying. To prevent this from happening we would like to present the user with multiple images (say, twenty) at each round of pool-querying. Thus, for each round, the active learner has to choose not just one image to be labeled but twenty. Theoretically it would be possible to consider the size of the resulting version spaces for each possible labeling of each possible set of twenty pool-queries but clearly this is impractical. Thus instead, for matters of computational efficiency, our SVM$_{Active}$ system takes the simple approach of choosing the pool-queries to be the twenty images closest to its separating hyperplane.

It has been noted [34] that the Simple querying algorithm used by SVM$_{Active}$ can be unstable during the first round of querying. To address this issue, SVM$_{Active}$ always randomly chooses twenty images for the first relevance feedback round. Then it uses the Simple active querying method on the second and subsequent rounds.

SVM$_{Active}$ Algorithm Summary

To summerize, our SVM$_{Active}$ system performs the following for each round of relevance feedback:

- Learn an SVM on the current labeled data
- If this is the first feedback round, ask the user to label twenty randomly selected images. Otherwise, ask the user to label the twenty pool images closest to the SVM boundary.

After the relevance feedback rounds have been performed SVM$_{Active}$ retrieves the top-k most relevant images:

- Learn a final SVM on the labeled data.
- The final SVM boundary separates "relevant" images from irrelevant ones. Display the k "relevant" images that are farthest from the SVM boundary.

5. IMAGE CHARACTERIZATION

We believe that image characterization should follow human perception [11]. In particular, our perception works in a multi-resolution fashion. For some visual tasks, our eyes may select coarse filters to obtain coarse image features; for others, they acquire finer features. Similarly, for some image applications (e.g., for detecting image replicas), employing coarse features is sufficient; for other applications (e.g., for classifying images), employing finer features may be essential. An image search engine thus must have the flexibility to model subjective perception and to satisfy a variety of search tasks.

Filter Name	Resolution	Representation
Masks	Coarse	Appearance of culture colors
Spread	Coarse	Spatial concentration of a color
Elongation	Coarse	Shape of a color
Histograms	Medium	Distribution of colors
Average	Medium	Similarity comparison within the same culture color
Variance	Fine	Similarity comparison within the same culture color

Table 1: Multi-resolution Color Features.

Our image retrieval system employs a multi-resolution image representation scheme [5]. In this scheme, we characterize images by two main features: color and texture. We consider shape as attributes of these main features.

5.1 Color

Although the wavelength of visible light ranges from 400 nanometers to 700 nanometers, research [11] shows that the colors that can be named by all cultures are generally limited to eleven. In addition to *black* and *white*, the discernible colors are *red*, *yellow*, *green*, *blue*, *brown*, *purple*, *pink*, *orange* and *gray*.

We first divide color into 12 color bins including 11 bins for culture colors and one bin for outliers [13]. At the coarsest resolution, we characterize color using a color mask of 12 bits. To record color information at finer resolutions, we record eight additional features for each color. These eight features are color histograms, color means (in H, S and V channels), color variances (in H, S and V channel), and two shape characteristics: elongation and spreadness. Color elongation characterizes the shape of a color and spreadness characterizes how that color scatters within the image [17]. Table 1 summarizes color features in coarse, medium and fine resolutions.

5.2 Texture

Texture is an important cue for image analysis. Studies [22, 32, 33, 21] have shown that characterizing texture features in terms of structuredness, orientation, and scale (coarseness) fits well with models of human perception. A wide variety of texture analysis methods have been proposed in the past. We choose a discrete wavelet transformation (DWT) using quadrature mirror filters [32] because of its computational efficiency.

Each wavelet decomposition on a 2-D image yields four subimages: a $\frac{1}{2} \times \frac{1}{2}$ scaled-down image of the input image and its wavelets in three orientations: horizontal, vertical and diagonal. Decomposing the scaled-down image further, we obtain the tree-structured or wavelet packet decomposition. The wavelet image decomposition provides a representation that is easy to interpret. Every subimage contains infor-

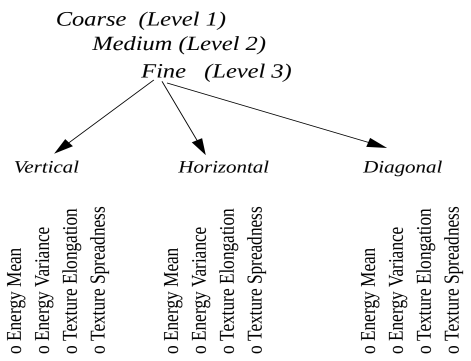


Figure 3: Multi-resolution Texture Features.

mation of a specific scale and orientation and also retains spatial information. We obtain nine texture combinations from subimages of three scales and three orientations. Since each subimage retains the spatial information of texture, we also compute elongation and spreadness for each texture channel. Figure 3 summarizes texture features.

Now, given an image, we can extract the above color and texture information to produce a 144 dimensional vector of numbers. We use this vector to represent the image. Thus, the space $\mathcal{X}$ for our SVMs is a 144 dimensional space, and each image in our database corresponds to a point in this space.

6. EXPERIMENTS

For our empirical evaluation of our learning methods we used three real-world image datasets: a four-category, a ten-category, and a fifteen-category image dataset where each category consisted of 100 to 150 images. These image datasets were collected from Corel Image CDs and the Internet.

- *Four-category* set. The 602 images in this dataset belong to four categories — *architecture*, *flowers*, *landscape*, and *people*.
- *Ten-category* set. The 1277 images in this dataset belong to ten categories — *architecture*, *bears*, *clouds*, *flowers*, *landscape*, *people*, *objectionable images*, *tigers*, *tools*, and *waves*. In this set, a few categories were added to increase learning difficulty. The tiger category contains images of tigers on landscape and water backgrounds to confuse with the landscape category. The objectionable images can be confused with people wearing little clothing. Clouds and waves have substantial color similarity.
- *Fifteen-category* set. In addition to the ten categories in the above dataset, the total of 1920 images in this dataset includes *elephants*, *fabrics*, *fireworks*, *food*, and *texture*. We added elephants with landscape and water background to increase learning difficulty between landscape, tigers and elephants. We added colorful fabrics and food to interfere with flowers. Various texture images (e.g., skin, brick, grass, water, etc.) were added to raise learning difficulty for all categories.

To enable an objective measure of performance, we assumed that a query concept was an image category. The SVM_{Active} learner has no prior knowledge about image categories[1]. It treats each image as a 144-dimension vector described in Section 5. The goal of SVM_{Active} is to learn a given concept through a relevance feedback process. In this process, at each feedback round SVM_{Active} selects twenty images to ask the user to label as "relevant" or "not relevant" with respect to the query concept. It then uses the labeled instances to successively refine the concept boundary. After the relevance feedback rounds have finished SVM_{Active} then retrieves the top-k most relevant images from the dataset based on the final concept it has learned. Accuracy is then computed by looking at the fraction of the k returned result that belongs to the target image category. Notice that this is equivalent to computing the precision on the top-k images. This measure of performance appears to be the most appropriate for the image retrieval task — particularly since, in most cases, not all of the relevant images will be able to be displayed to the user on one screen. As in the case of web searching, we typically wish the first screen of returned images to contain a high proportion of relevant images. We are less concerned that not every single instance that satisfies the query concept is displayed. As with all SVM algorithms, SVM_{Active} requires at least one relevant and one irrelevant image to function. In practice a single relevant image could be provided by the user (e.g., via an upload to the system) or could be found by displaying a large number of randomly selected images to the user (where, perhaps, the image feature vectors are chosen to be mutually distant from each other so as to provide a wide coverage of the image space). In either case we assume that we start off with one randomly selected relevant image and one randomly selected irrelevant image.

6.1 SVM_{Active} Experiments

Figures 4(a-c) show the average top-k accuracy for the three different sizes of data sets. We considered the performance of SVM_{Active} after each round of relevance feedback. The graphs indicate that performance clearly increases after each round. Also, the SVM_{Active} algorithm's performance degrades gracefully when the size and complexity of the database is increased – for example, after four rounds of relevance feedback it achieves an average of 100%, 95%, 88% accuracy on the top-20 results for the three different sizes of data sets respectively. It is also interesting to note that SVM_{Active} is not only good at retrieving just the top few images with high precision, but it also manages to sustain fairly high accuracy even when asked to return larger numbers of images. For example, after five rounds of querying it attains 99%, 84% and 76% accuracy on the top-70 results for the three different sizes of data sets respectively. SVM_{Active} uses the active querying method outlined in Section 4. We examined the effect that the active querying method had on performance. Figures 5(a) and 5(b) compare the active querying method with the regular passive method of sampling. The passive

[1]Unlike some recently developed systems [37] that contain a semantical layer between image features and queries to assist query refinement, our system does not have an explicit semantical layer. We argue that having a hard-coded semantical layer can make a retrieval system restrictive. Rather, dynamically learning the semantics of a query concept is more flexible and hence makes the system more useful.

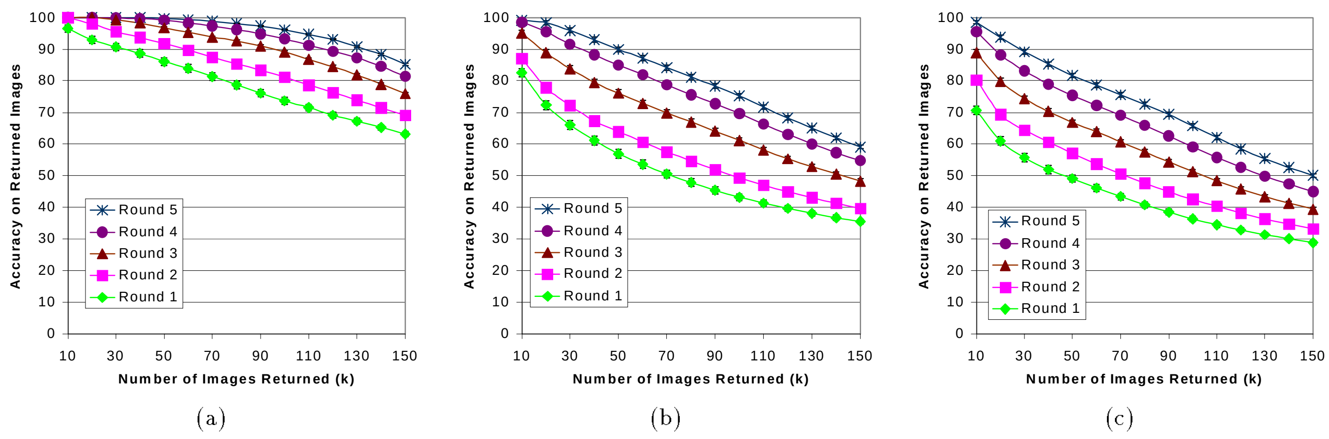


Figure 4: (a) Average top-k accuracy over the four-category dataset. (b) Average top-k accuracy over the ten-category dataset. (c) Average top-k accuracy over the fifteen-category dataset. Standard error bars are smaller than the curves' symbol size. Legend order reflects order of curves.

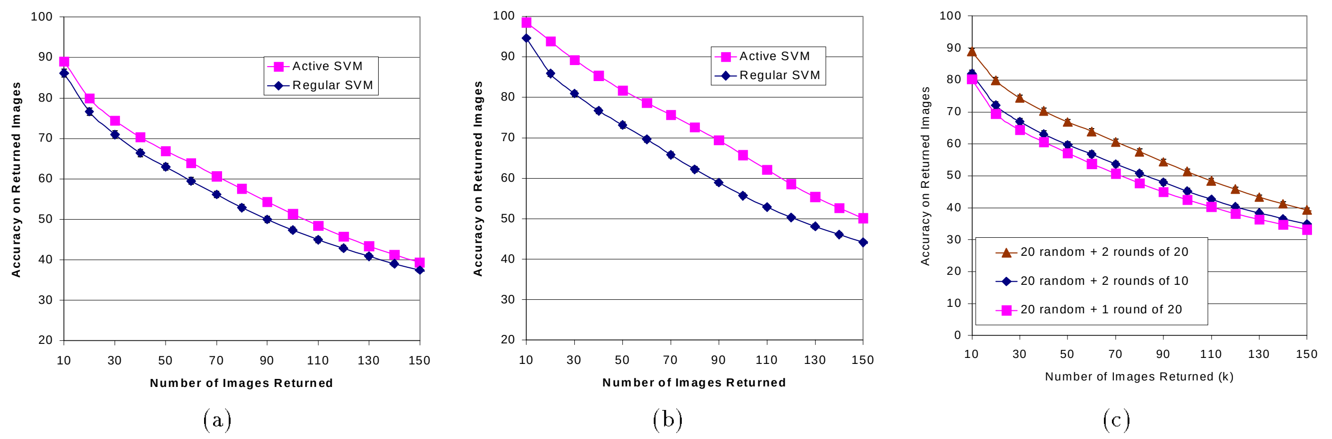
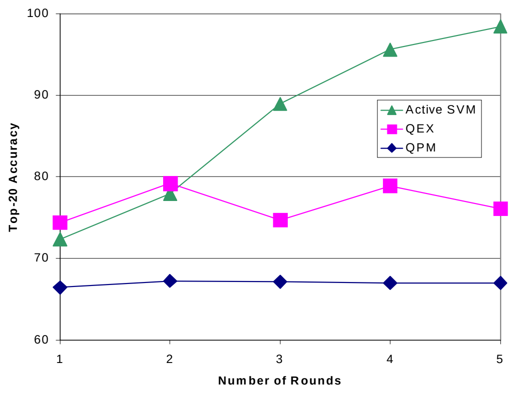
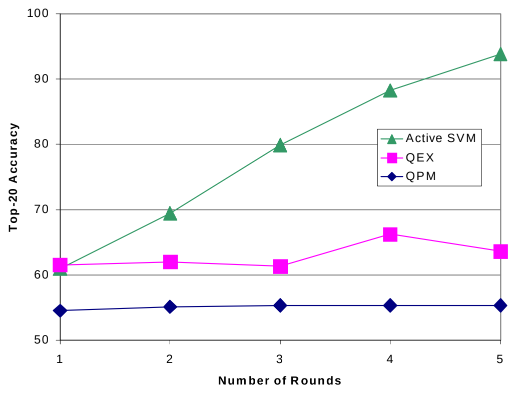


Figure 5: (a) Active and regular passive learning on the fifteen-category dataset after three rounds of querying. (b) Active and regular passive learning on the fifteen-category dataset after five rounds of querying. (c) Comparison between asking ten images per pool-query round and twenty images per pool-querying round on the fifteen-category dataset. Standard error bars are smaller than the curves' symbol size. Legend order reflects order of curves.

method chooses random images from the pool to be labeled. This method is the one that is typically used with SVMs since it creates a randomly selected data set. It is clear that the use of active learning is beneficial in the image retrieval domain. There is a significant increase in performance from using the active method. SVM$_{Active}$ displays 20 images per pool-querying round. There is a tradeoff between the number of images to be displayed in one round, and the number of querying rounds. The fewer images displayed per round, the lower the performance. However, with fewer images per round we may be able to conduct more rounds of querying and thus increase our performance. Figure 5(c) considers the effect of displaying only ten images per round. In this experiment our first round consisted of displaying twenty random images and then, on the second and subsequent rounds of querying, active learning with 10 or 20 images is invoked. We notice that there is indeed a benefit to asking (20 random + two rounds of 10 images) over asking (20 random + one round of 20 images). This is unsurprising since the active learner has more control and freedom to adapt when asking two rounds of 10 images rather than one round of 20. What is interesting is that asking (20 random + two rounds of 20 images) is far, far better than asking (20 random + two rounds of 10 images). The increase in the cost to users of asking 20 images per round is often negligible since users can pick out relevant images easily. Furthermore, there is virtually no additional computational cost in calculating the 20 images to query over the 10 images to query. Thus, for this particular task, we believe that it is worthwhile to display around 20 images per screen and limit the number of querying rounds, rather than display fewer images per screen and use many more querying rounds. We also investigated how performance altered when various aspects of the algorithm were changed. Table 2 shows how all three of the texture resolutions are important. Also, the performance of the SVM appears to be greatest when all of the texture resolutions are included (although in this case the difference is not statistically significant). Table 3 indicates how other



(a)



(b)

Figure 6: (a) Average top-k accuracy over the ten-category dataset. (b) Average top-k accuracy over the fifteen-category dataset.

Texture features	Top-50 Accuracy
None	80.6 ± 2.3
Fine	85.9 ± 1.7
Medium	84.7 ± 1.6
Coarse	85.8 ± 1.3
All	86.3 ± 1.8

Table 2: Average top-50 accuracy over the four-category data set using a regular SVM trained on 30 images. Texture spatial features were omitted.

	Top-50	Top-100	Top-150
Degree 2 Polynomial	95.9 ± 0.4	86.1 ± 0.5	72.8 ± 0.4
Degree 4 Polynomial	92.7 ± 0.6	82.8 ± 0.6	69.0 ± 0.5
Radial Basis	$\mathbf{96.8 \pm 0.3}$	$\mathbf{89.1 \pm 0.4}$	$\mathbf{76.0 \pm 0.4}$

Table 3: Accuracy on four-category data set after three querying rounds using various kernels. Bold type indicates statistically significant results.

SVM kernel functions perform on the image retrieval task compared to the radial basis function kernel. It appears that the radial basis function kernel is the most suitable for this feature space. One other important aspect of any relevance feedback algorithm is the wall clock time that it takes to generate the next pool-queries. Relevance feedback is an interactive task, and if the algorithm takes too long then the user is likely to lose patience and be less satisfied with the experience. Table 4 shows that SVM$_{Active}$ averages about a second on a Sun Workstation to determine the 20 most informative images for the users to label. Retrieval of the 150 most relevant images takes an similar amount of time and computing the final SVM model never exceeds two seconds.

Dataset	Dataset Size	round of 20 queries (secs)	Computing final SVM	Retrieving top 150 images
4 Cat	602	0.34 ± 0.00	0.5 ± 0.01	0.43 ± 0.02
10 Cat	1277	0.71 ± 0.01	1.03 ± 0.03	0.93 ± 0.03
15 Cat	1920	1.09 ± 0.02	1.74 ± 0.05	1.37 ± 0.04

Table 4: Average run times in seconds

6.2 Scheme Comparison

We also compared SVM$_{Active}$ with two traditional query refinement methods: *query point movement* (QPM) and *query expansion* (QEX). In this experiment, each scheme returned the 20 most relevant images after up to five rounds of relevance feedback. To ensure that the comparison to SVM$_{Active}$ was fair, we seeded both schemes with one randomly selected relevant image to generate the first round of images. On the ten-category image dataset, Figure 6(a) shows that SVM$_{Active}$ achieves nearly 90% accuracy on the top-20 results after three rounds of relevance feedback, whereas the accuracies of both QPM and QEX never reach 80%. On the fifteen-image category dataset, Figure 6(b) shows that SVM$_{Active}$ outperforms the others by even wider margins. SVM$_{Active}$ reaches 80% top-20 accuracy after three rounds and 94% after five rounds, whereas QPM and QEX cannot achieve 65% accuracy.

These results hardly surprise us. Traditional information retrieval schemes require a large number of image instances to achieve any substantial refinement. By refined current relevant instances both QPM and QEX tend to be fairly localized in their exploration of the image space and hence rather slow in exploring the entire space. During the relevance feedback phase SVM$_{Active}$ takes both the relevant and irrelevant images into account when choosing the next pool-queries. Furthermore, it chooses to ask the user to label images that it regards as most **informative** for learning the query concept, rather than those that have the most likelihood of being relevant. Thus it tends to explore the feature space more aggressively.

Figures 7 and 8 show an example run of the SVM$_{Active}$ system. For this run, we are interested in obtaining architecture images. In Figure 7 we initialize the search by giving SVM$_{Active}$ one relevant and one irrelevant image. We then have three feedback rounds. The images that SVM$_{Active}$ asks us to label in these three feedback rounds are images that SVM$_{Active}$ will find most informative to know about. For example, we see that it asks us to label a number of landscape images and other images with a blue or gray background with something in the foreground. The feedback rounds allow SVM$_{Active}$ to narrow down the types of

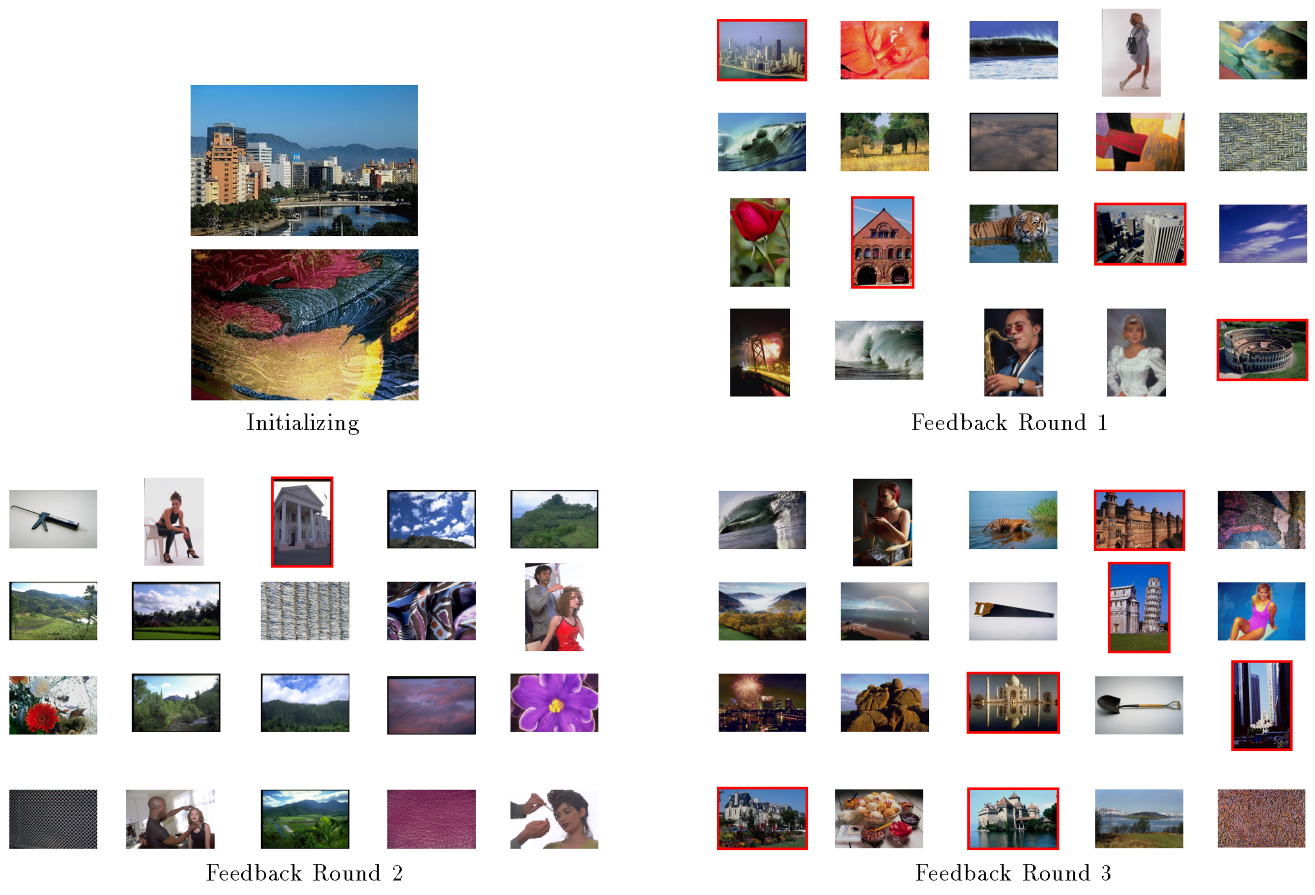


Figure 7: Searching for architecture images. SVM$_{Active}$ Feedback phase.

images that we like. When it comes to the retrieval phase (Figure 7) SVM$_{Active}$ returns, with high precision, a large varitey of different architecture images, ranging from old buildings to modern cityscapes.

7. RELATED WORK

There have been several studies of active learning for classification in the machine learning community. The *query by committee* algorithm [30, 10] uses a distribution over all possible classifiers and attempts to greedily reduce the entropy of this distribution. This general purpose algorithm has been applied in a number of domains[2] using classifiers (such as Naive Bayes classifiers [8, 23]) for which specifying and sampling classifiers from a distribution is natural. Probabilistic models such as the Naive Bayes classifier provide interpretable models and principled ways to incorporate prior knowledge and data with missing values. However, they typically do not perform as well as discriminative methods such as SVMs [15, 9].

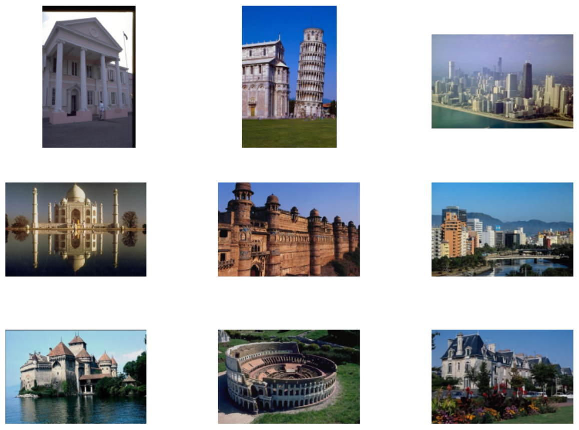
Lewis and Gale (1994) focused on the text classification task and introduced uncertainty sampling using logistic regression and also decision trees [18]. SVM$_{Active}$'s querying method is essentially the same as their uncertainty sampling method (choose the instance that our current classifier is most uncertain about).

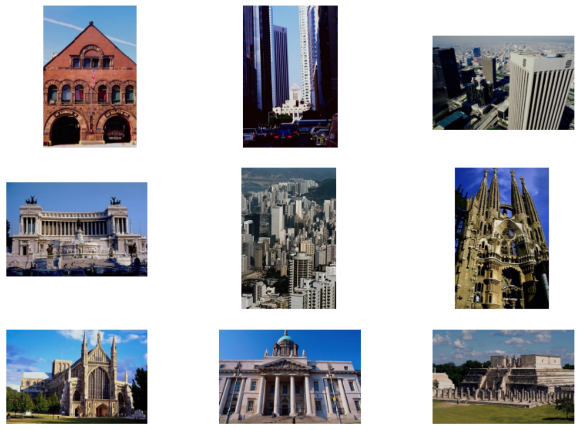
[2]Although, to our knowledge, not to the image retrieval domain.

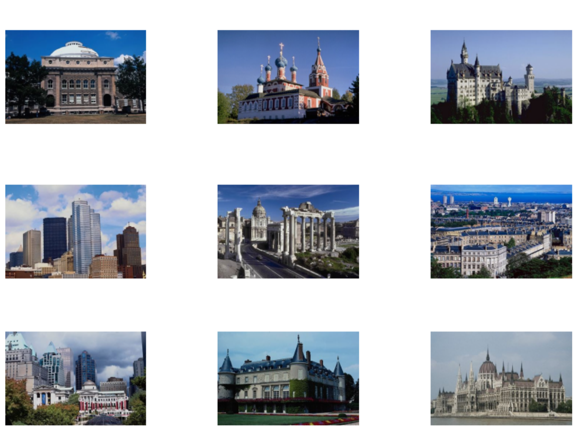
Relevance feedback techniques proposed by the information retrieval and database communities also perform non-random sampling. The study of [28] puts these query refinement approaches into three categories: *query reweighting*, *query point movement* and *query expansion*.

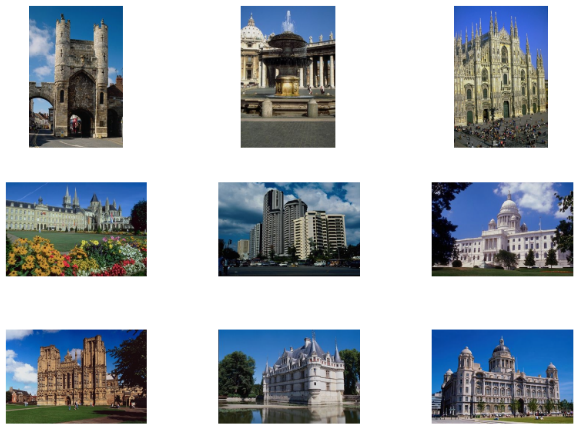
- *Query reweighting* and *query point movement* [14, 25, 27]. Both query reweighting and query point movement use nearest-neighbor sampling: They return top ranked objects to be marked by the user and refine the query based on the feedback. These methods tend to require a large number of image instances to achieve any substantial refinement since they do not tend to explore the image space aggressively.
- *Query expansion* [28, 38]. The *query expansion* approach can be regarded as a multiple-instances sampling approach. The samples of the next round are selected from the neighborhood (not necessarily the nearest ones) of the positive-labeled instances of the previous round. The study of [28] shows that query expansion achieves only a slim margin of improvement (about 10% in precision/recall) over query point movement.

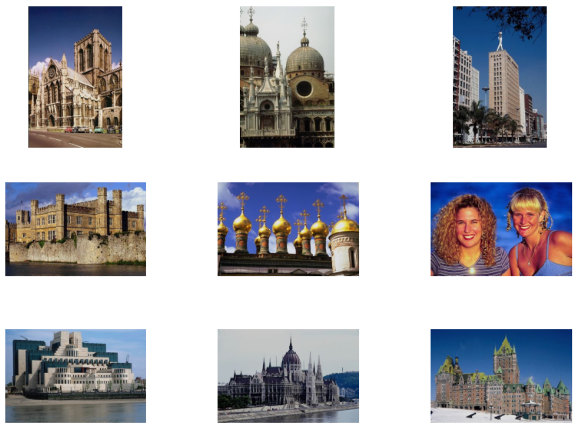
For image retrieval, the PicHunter system [7] uses Bayesian prediction to infer the goal image, based upon the users'


First Screen of Results


Second Screen of Results


Third Screen of Results


Fourth Screen of Results


Fifth Screen of Results

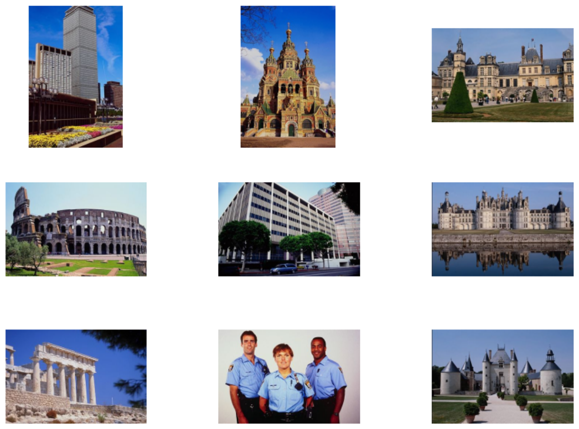

Sixth Screen of Results

Figure 8: Searching for architecture images. SVM_{Active} **Retrieval phase.**

actions. The system shows that employing active learning can drastically cut down the number of iterations (up to 80% in some experiments). However the authors also point out that their scheme is computationally intensive, since it needs to recompute the conditional probability for all unlabeled samples after each round of user feedback and hence may not scale well with dataset size.

8. CONCLUSIONS AND FUTURE WORK

We have demonstrated that active learning with support vector machines can provide a powerful tool for searching image databases, outperforming a number of traditional query refinement schemes. SVM_{Active} not only achieves consistently high accuracy on a wide variety of desired returned results, but also does it quickly and maintains high precision when asked to deliver large quantities of images. Also, unlike recent systems such as SIMPLIcity [37], it does not require an explicit semantical layer to perform well.

There are a number of interesting directions that we wish to pursue. The running time of our algorithm scales linearly with the size of the image database both for the relevance feedback phase and for the retrieval of the top-k images. This is because, for each querying round, we have to scan through the database for the twenty images that are closest to the current SVM boundary, and in the retrieval phase we have to scan the entire database for the top k most relevant images with respect to the learned concept. SVM_{Active} is practical for image databases that contain a few thousand images; however, we would like to find ways for it to scale to larger sized databases.

For the relevance feedback phase, one possible way to cope with a large image database is, rather than using the entire database as the pool, instead sample a few thousand images from the database and use these as the pool of potential images with which to query the user. The technique of subsampling databases is often used effectively when performing data mining with large databases (e.g., [6]). It is plausible that this technique will have a negligible effect on overall accuracy, while significantly speeding up the running time of the SVM_{Active} algorithm on large databases. Retrieval speed of relevant images in large databases can perhaps be sped up significantly by using intelligent clustering and indexing schemes [20].

The second direction we wish to pursue is an issue that faces many relevance feedback algorithms: that of designing methods to seed the algorithm effectively. At the moment we assume that we are presented with one relevant image and one irrelevant image. It would be beneficial to modify SVM_{Active} so that it is not dependent on having a relevant starting image. We are currently investigating ways of using SVM_{Active}'s output to explore the feature space effectively until a single relevant image is found.

An alternative approach for finding a single relevant image is to use another algorithm to seed SVM_{Active}. For example, the MEGA algorithm [4] that we have developed in a separate study does not require seeding with a relevant image. If all of the images generated in the first round of its relevance feedback are irrelevant, it will use the negative images to reduce the set of potentially relevant images substantially so that a relevant image will be generated in the next round with high probability. Indeed, preliminary experiments that use MEGA to find and initial relevant image, and SVM_{Active} to perform relevance feedback have produced promising results [3].

Transduction takes advantage of not only the labeled images but also the unlabeled images to learn a user's query concept. Intuitively, the unlabeled images are likely to be clustered and this provides us with some extra information which we may be able to use to refine our idea of what the user's query concept may be. Transduction has been successfully applied to regular passive SVMs for text classification [16] showing that using unlabeled data can indeeed improve performance. An exciting direction is to see whether transduction can be applied to the image retrieval domain and furthermore, whether it can also be combined with active learning to provide yet further increases in performance for the image retrieval task.

9. ACKNOWLEDGEMENTS

Simon Tong was supported by DARPA's *Information Assurance* program under subcontract to SRI International, and by ONR MURI N00014-00-1-0637.

10. REFERENCES

[1] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[2] C. Campbell, N. Cristianini, and A. Smola. Query learning with large margin classifiers. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

[3] E. Chang and et al. PBIR — a system that learns subjective image query concepts. *ACM Multimedia (Demo)*, October 2001.

[4] E. Chang and B. Li. Mega — the maximizing expected generalization algorithm for learning complex query concepts (extended version). *Technical Report http://www-db.stanford.edu/∼echang/mega.ps*, November 2000.

[5] E. Chang, B. Li, and C. Li. Towards perception-based image retrieval. *IEEE Content-Based Access of Image and Video Libraries*, pages 101–105, June 2000.

[6] S. Chaudhuri, V. Narasayya, and R. Motwani. Random sampling for histogram construction: How much is enough? *ACM Sigmod*, May 1998.

[7] I. J. Cox, M. L. Miller, T. P. Minka, T. V. Papathomas, and P. N. Yianilos. The bayesian image retrieval system, pichunter: Theory, implementation and psychological experiments. *IEEE Transaction on Image Processing (to appear)*, 2000.

[8] I. Dagan and S. Engelson. Committee-based sampling for training probabilistic classifiers. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 150–157. Morgan Kaufmann, 1995.

[9] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the Seventh International Conference on Information and Knowledge Management.* ACM Press, 1998.

[10] Y. Freund, H. Seung, E. Shamir, and N. Tishby. Selective sampling using the Query by Committee algorithm. *Machine Learning*, 28:133–168, 1997.

[11] E. B. Goldstein. *Sensation and Perception (5^{th} Edition).* Brooks/Cole, 1999.

[12] R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines: Estimating the bayes point in kernel space. In *International Joint Conference on Artificial Intelligence Workshop on Support Vector Machines*, pages 23–27, 1999.

[13] K. A. Hua, K. Vu, and J.-H. Oh. Sammatch: A flexible and efficient sampling-based image retrieval technique for image databases. *Proceedings of ACM Multimedia*, November 1999.

[14] Y. Ishikawa, R. Subramanya, and C. Faloutsos. Mindreader: Querying databases through multiple examples. *VLDB*, 1998.

[15] T. Joachims. Text categorization with support vector machines. In *Proceedings of the European Conference on Machine Learning*. Springer-Verlag, 1998.

[16] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 200–209. Morgan Kaufmann, 1999.

[17] J.-G. Leu. Computing a shape's moments from its boundary. *Pattern Recognition*, pages Vol.24, No.10,pp.949–957, 1991.

[18] D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1994.

[19] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12. Springer-Verlag, 1994.

[20] C. Li, E. Chang, H. Garcia-Molina, and G. Wiederhold. Clustering for approximate similarity queries in high-dimensional spaces. *IEEE Transaction on Knowledge and Data Engineering (to appear)*, 2001.

[21] W. Y. Ma and H. Zhang. Benchmarking of image features for content-based retrieval. *Proceedings of Asilomar Conference on Signal, Systems & Computers*, 1998.

[22] B. Manjunath, P. Wu, S. Newsam, and H. Shin. A texture descriptor for browsing and similarity retrieval. *Signal Processing Image Communication*, 2001.

[23] A. McCallum and K. Nigam. Employing EM in pool-based active learning for text classification. In *Proceedings of the Fifteenth International Conference on Machine Learning.* Morgan Kaufmann, 1998.

[24] T. Mitchell. Generalization as search. *Artificial Intelligence*, 28:203–226, 1982.

[25] M. Ortega, Y. Rui, K. Chakrabarti, A. Warshavsky, S. Mehrotra, and T. S. Huang. Supporting ranked boolean similarity queries in mars. *IEEE Transaction on Knowledge and Data Engineering*, 10(6):905–925, December 1999.

[26] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *Proceedings of the International Conference on Computer Vision*, 1998.

[27] K. Porkaew, K. Chakrabarti, and S. Mehrotra. Query refinement for multimedia similarity retrieval in mars. *Proceedings of ACM Multimedia*, November 1999.

[28] K. Porkaew, S. Mehrota, and M. Ortega. Query reformulation for content based multimedia retrieval in mars. *ICMCS*, pages 747–751, 1999.

[29] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

[30] H. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Workshop on Computational Learning Theory*, pages 287–294. Morgan Kaufmann, 1992.

[31] J. Shawe-Taylor and N. Cristianini. Further results on the margin distribution. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 278–285, 1999.

[32] J. Smith and S.-F. Chang. Automated image retrieval using color and texture. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, November 1996.

[33] H. Tamura, S. Mori, and T. Yamawaki. Texture features corresponding to visual perception. *IEEE Transaction on Systems Man Cybernet (SMC)*, 1978.

[34] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Proceedings of the 17th International Conference on Machine Learning*, pages 401–412, June 2000.

[35] V. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer Verlag, 1982.

[36] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[37] J. Wang, J. Li, and G. Wiederhold. Simplicity: Semantics-sensitive integrated matching for picture libraries. *ACM Multimedia Conference*, 2000.

[38] L. Wu, C. Faloutsos, K. Sycara, and T. R. Payne. Falcon: Feedback adaptive loop for content-based retrieval. *The 26^{th} VLDB Conference*, September 2000.