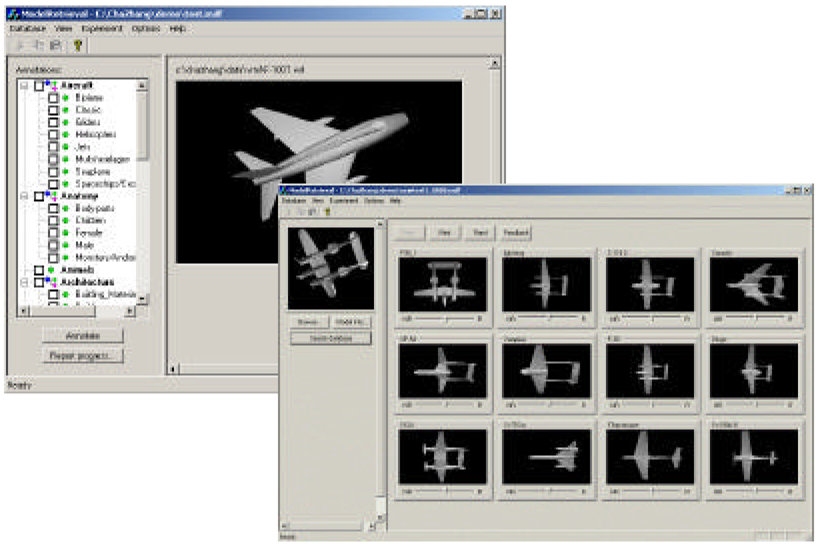
An Active Learning Framework for Content Based Information Retrieval

Cha Zhang and Tsuhan Chen
Advanced Multimedia Processing Lab



Technical Report AMP 01-04
Revised Mar. 2002

Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In this paper, we propose a general active learning framework for content-based information retrieval. We use this framework to guide hidden annotations in order to improve the retrieval performance. For each object in the database, we maintain a list of probabilities, each indicating the probability of this object having one of the attributes. During training, the learning algorithm samples objects in the database and presents them to the annotator to assign attributes to. For each sampled object, each probability is set to be one or zero depending on whether or not the corresponding attribute is assigned by the annotator. For objects that have not been annotated, the learning algorithm estimates their probabilities with kernel regression. Furthermore, the normal kernel regression algorithm is modified into a biased kernel regression, so that an object that is far from any annotated object will receive an estimate result of the prior probability. This is based on our basic assumption that any annotation should not propagate too far in the feature space if we cannot guarantee that the feature space is good. *Knowledge gain* is then defined to determine, among the objects that have not been annotated, which one the system is the most uncertain of, and present it as the next sample to the annotator to assign attributes to. During retrieval, the list of probabilities works as a feature vector for us to calculate the semantic distance between two objects, or between the user query and an object in the database. The overall distance between two objects is determined by a weighted sum of the semantic distance and the low-level feature distance. The algorithm is tested on both synthetic database and real database. In both cases the retrieval performance of the system improves rapidly with the number of annotated samples. Furthermore, we show that active learning outperforms learning based on random sampling.

Key words: active learning, content-based information retrieval, attribute tree, biased kernel regression, semantics.

. Contact Author: Prof. Tsuhan Chen, Department of Electrical and Computer Engineering, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213. Tel: (412) 268-7536, Fax: (412) 268-3890, Email: tsuhan@cmu.edu
. Work supported in part by NSF CAREER Award 9984858.
. To appear in *IEEE Trans. on Multimedia, Special Issue on Multimedia Database*, June 2002.

I. Introduction

Content-based information retrieval (CBIR) has attracted a lot of research interest in recent years. A typical CBIR system, e.g., an image retrieval system, includes three major aspects: feature extraction, high dimensional indexing, and system design [1]. Among the three aspects, feature extraction is the basis of content-based information retrieval. However, features we can extract from the data are often low-level features. We call them low-level features because most of them are extracted directly from digital representations of objects in the database and have little or nothing to do with how a human would perceive or recognize them. As a result, two semantically similar objects may lie far from each other in the feature space, while two completely different objects may stay close to each other in the same space. Although many features have been designed for general or specific CBIR systems, and some of them showed good retrieval performance, the gap between low-level features and high-level semantic meanings of the objects has been the major obstacle to more successful retrieval performance.

Relevance feedback and hidden annotation have been shown to be two of the most powerful tools for bridging the gap between low-level features and high-level semantics. Widely used in text retrieval [2][3], relevance feedback was first proposed by Rui *et al.* as an interactive tool in content-based image retrieval [4]. Since then it has been proven to be a powerful tool and has become a major focus of research in this area [5][6][7][8][9][10]. In *MindReader*, Ishikawa *et al.* formulated a minimization problem on the ideal query parameter estimation process [5]. Rui and Huang improved the approach further and took into consideration the multi-level image model [6]. A novel approach was proposed by Tian *et al.* to provide both positive and negative feedback for learning with Support Vector Machines (SVM) [7]. Sull *et al.* presented a framework for accumulating image relevance information through relevance feedback and constructing a relevance graph for later usage [8]. Nonlinear feedback was performed by Nikolaos, *et al.* [9], where an adaptively trained neural network architecture is adopted to realize the nonlinear feedback. Furthermore, Minka and Picard [10] showed the possibility to construct new features "on the fly" during the interaction between the user and the system.

Relevance feedback moves the query point towards the relevant objects or selectively weighs the features in the low-level feature space based on user feedback. It does not take into account the actual semantics of the objects themselves. The assumption is that the low-level feature space is complete enough to represent high-level semantics, so we can achieve good results simply by changing the weights of the features or by moving the query point. Unfortunately, this assumption is not necessarily true. In many cases, low-level features are unable to describe high-level semantics. As an example, if the low-level features of a set of semantically similar objects lie in the space as several clusters, querying with an object in one cluster would not be able to retrieve semantically similar objects in other clusters by reweighing the space. In [12] and [13], similar approaches were proposed to use relevance feedback to build semantic relationships inside the database. Their systems grouped the objects in the database into small semantic clusters and related the clusters with semantic weights. The updating of the clusters and semantic weights are based on the user's feedback. Another solution to the above problem is hidden annotation. By attaching Boolean attributes to images in the database, Cox *et al.* did some experiments on hidden annotation in their Bayesian image retrieval system, *PicHunter*, and showed positive results [11]. In this paper, we study the hidden annotation as a preprocessing stage of a retrieval system, referred to as the learning stage, before any user can use the system.

The first observation we have is that it is better to view the hidden annotation and the relevance feedback as two separate stages in a retrieval system, although they are able to work together. When a user is using the system to retrieve some results with a query, the similarity measurement in the user's mind might be changing all the time. This is why most of the relevance feedback systems do not keep the user's previous feedback for later queries [4]. For the same reason, it is not reasonable to assume that the user's feedback can always be used to update the hidden annotation consistently. For example, sometimes the user may want to find something that is round in the database, and the user does not care if it is a ball or an apple. An annotation-update based on this may deteriorate the system performance when the later the same user, or other users, may want to find a ball instead of an apple. In [12], Lu *et al.* tried to solve this problem by slowly increasing and dramatically decreasing the weights of the links between images and keywords. In this paper, we decide to make hidden annotation a preprocessing stage, referred to as the learning stage, before any user can use the system. Learning similarities before retrieval is not a new idea. Ma and Manjunath [14] used a hybrid neural network to learn the similarities between objects by clustering them in the low-level feature space. Parts of the data were used for training, and the others were used for testing.

The second observation we have is that most of existing systems using hidden annotation either annotate all the objects in the database (full annotation), or annotate a subset of the database manually selected (partial annotation). As the database becomes larger, full annotation is increasingly difficult because of the manual effort involved. Partial annotation is relatively affordable and trims down the heavy manual labor. Once the database is partially annotated, traditional pattern classification methods are often used to derive semantics of the objects not yet annotated. However, it is not clear how much annotation is sufficient for a specific database, and what the best subset of objects to annotate is. In this paper, we use active learning to determine which objects should be annotated. During the learning stage, the system provides sample objects automatically to the annotator. The sample objects are selected based on how much information annotation of each sample object can provide to decrease the *uncertainty* of the system. The object, once annotated, gives the maximum information or knowledge gain to the system is selected. In the machine learning literature, the idea of maximizing the expected information from a query has been studied under the name "active learning" or "learning with queries" [15]. It was revisited by Cox *et al.* when they updated the display of the query result in [11]. We will present a more detailed survey of the active learning literature in Section II-B.

The key assumption we make throughout this paper is that, although the low-level feature space cannot describe the semantic meaning, it is *locally inferable*. This means that in the low-level feature space, if two objects are very close to each other, they should be semantically similar, or be able to infer some knowledge to each other. On the other hand, if two objects are far from each other, the semantic link between them should be weak. Notice that because of the locality of the semantic inference, this assumption allows objects with the same semantic meaning to lie in different places in the feature space, which cannot be handled by normal relevance feedback. We argue that if the above assumption does not hold, neither relevance feedback nor hidden annotation will be able to help improving the retrieval performance, even the database is fully annotated. The only solution to this circumstance might be to find better low-level features for the objects.

We assume that the semantic meanings of the objects in the database can be characterized by a multi-level attribute tree. To make the attribute tree general, the attributes at the same level of the tree are not necessarily exclusive of each other. For each object in the database, we maintain a list of probabilities, each of them indicating the probability of this object having the corresponding attribute. If an object is annotated, the probabilities are set to be one or zero depending on whether the corresponding attributes are annotated to characterize the object or not. For each of the objects that have not been annotated, we estimate its attribute probabilities based on its annotated neighbors. Kernel regression is employed to fulfill this task. With this list of probabilities, we are able to tell which object the system is most uncertain of, and propose it as a sample to the annotator. The list of probabilities also works as a feature vector to calculate the semantic distance between two objects. The final similarity measurement between any two objects is determined by a weighted sum of the semantic distance and the low-level feature distance. We show that with our algorithm, the performance of the retrieval system improves rapidly with the number of annotated models, and in all cases outperforms the approach of randomly choosing the objects to annotate.

The paper is organized as follows. In Section II, we introduce the general criterion for active learning in our approach. Section III presents the details of the proposed algorithm. We discuss the joint-semantic-low-level feature similarity measurement in Section IV. We show the experimental results in Section V and conclude the paper in Section VI.

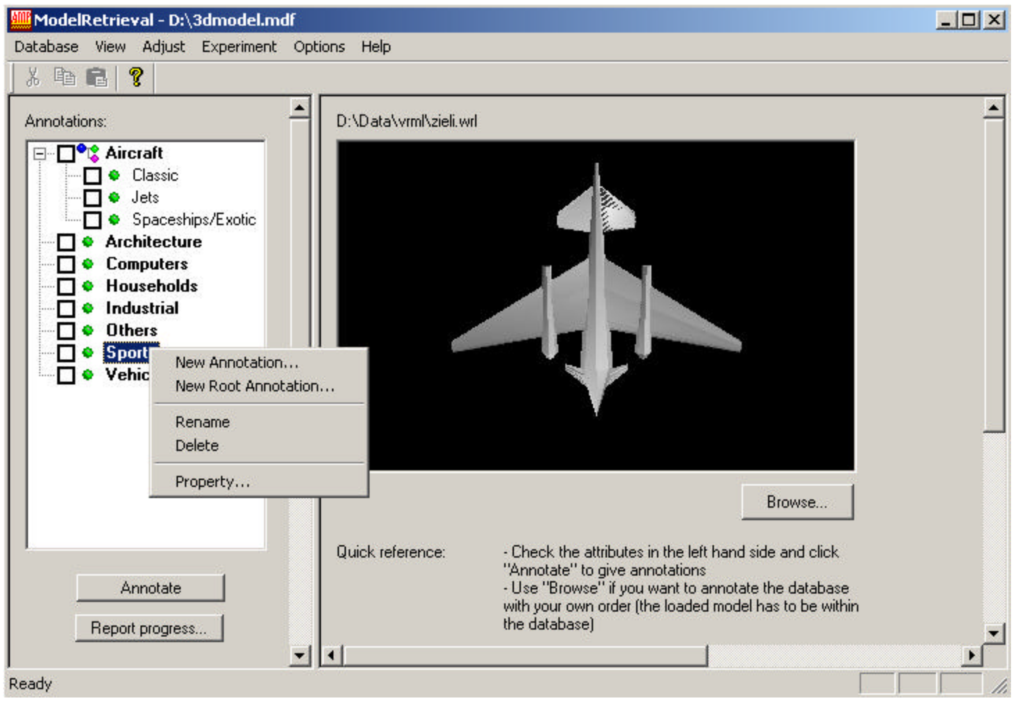
II. The General Criterion for Active Learning

In this section, we set up the active learning problem in a content-based information retrieval system and discuss the general criterion for active learning. We show our learning interface and define the attribute tree structure in Section II-A. In Section II-B, a brief introduction of the active learning literature is provided followed by the general criterion we use to choose the next sample object.

A. The Learning Interface and the Attribute Tree Structure

Figure 1 shows the learning/annotation interface of our system. On the left hand side is a list of attributes to be annotated. On the right hand side is a sample object (e.g., an image or a 3D model) the system proposes. The basic operation for the annotator is to check some of the attributes for this sample model and press the "Annotate" button for the system to get the annotation information of the sample model.

In our system, the attributes form a tree structure with multiple levels. In the attribute tree, each node is an attribute. The attributes at higher-level nodes are more general than those at the lower-level nodes. By default we assume that once an attribute at a lower-level nodes is checked, the attributes at the higher-level nodes or its parent nodes are also checked. As a simple example, "Aircraft" lies at the first level that is the highest level in the tree structure, thus it is more general than "Jets", which lies at the second level. An object that is a "Jets" is also an "Aircraft". Unlike the decision tree in classification applications, the nodes with the same parent node in our attribute tree are not necessarily exclusive of each other. For example, an aircraft can be both a "Classic" and a "Jets". This makes our tree structure more general and more natural to use for annotation.



Quick reference: - Check the attributes in the left hand side and click "Annotate" to give annotations
- Use "Browse" if you want to annotate the database with your own order (the loaded model has to be within the database)

Figure 1 The learning interface and the tree annotation structure.

The annotation starts with no attributes in the tree structure. When necessary, the annotator may add, rename or remove any attributes at any level. The annotator is asked to check all the attributes that the sample object has. For an attribute that the annotator does not check, we assume the annotator implies that this object does not have that attribute, unless the attribute is the parent of another checked attribute.

B. The Literature of Active Learning and the General Criterion to Choose the Samples

For many types of machine learning algorithms, one can find the statistically "optimal" way to select the training data. The pursuing of the "optimal" way by the machine itself was referred to as *active learning*. While in traditional machine learning research, the learner typically works as a passive recipient of the data, active learning enables the learner to use its own ability to respond to collect data and to influence the world it is trying to understand. Some representative work on active learning can be found in [28][29][30].

To be more specific, what we are interested is a specific form of active learning, i.e., *selective sampling*. The goal of selective sampling is to reduce the number of training samples that need to be annotated by examining objects that are not yet annotated and selecting the most informative ones for the annotator. Many approaches have been proposed for selective sampling. In [22], Seung *et al*. proposed an algorithm called *query by committee* (QBC). Query by committee generates a committee of classifiers and the next query is chosen by the principle of maximal disagreement among these classifiers. Freund *et al.* extended the QBC result to a wide range of classifier forms. They gave some theoretical proofs that, under some assumptions, the effect of training on annotated data can be achieved for the cost of obtaining data that are note yet annotated, and labeling only a logarithmic fraction of them [23]. In [24], Nigam and McCallum modified the QBC algorithm by a combination of active learning and the traditional Expectation-Maximization (EM) algorithm. The QBC algorithm assumes that the data is noise free, a perfect deterministic classifier exists, and all the classifiers in the committee agree with each other after full annotation. In a real world case, these assumptions are usually not true, and the effectiveness of QBC is not clear. In [27], Muslea *et al.* introduced an algorithm called *co-testing*. It is similar to the QBC algorithm and is designed to apply to problems with redundant views or problems with multiple disjoint sets of attributes (features) that can be used to learn the target attribute. Lewis and Gale [25] described in their paper another approach called *uncertainty sampling*. The idea is to use only one classifier not only tells which class a sample is, but also gives an uncertainty score for each data sample not yet annotated. The next sample is chosen based on which one the classifier has the least confident with. With uncertainty sampling, it was reported [25][26] that the size of the training data could be reduced as much as 500-fold for text classification.

We need to find a general criterion to measure how much information the annotator's annotation can provide to the system. Let $O_i, i = 1, 2, \ldots, N$ be the objects in the database, and $A_k, k = 1, 2, \ldots, K$ be the K attributes the annotator wants to use for annotation. These attributes form the whole attribute tree. For each object O_i, we define probability P_{ik} to be the probability that this object has attribute A_k, where $P_{ik} = 1$ means that the object O_i has been annotated as having attribute A_k, and $P_{ik} = 0$ means it has been annotated as not having attribute A_k. If the object has not been annotated, P_{ik} is estimated by its neighboring annotated objects, as will be described in Section III-B. In order to derive the expected information gain when we annotate a certain object, we define an uncertainty measurement as follows:

$$U_i = \Psi(P_{i1}, P_{i2}, \ldots, P_{iK}), \qquad i = 1, 2, \ldots, N \tag{1}$$

where U_i is the uncertainty measurement, $\Psi(\cdot)$ is a function on all the attribute probabilities of object O_i. We want the uncertainty measurement U_i to have the following properties:

1. If object O_i has been annotated, $U_i = 0$;
2. If $P_{ik} = 0.5$, for $k = 1, 2, \ldots, K$, i.e., we know nothing about the object, $U_i = U_{max}$;
3. Given $P_{ik}, k = 1, 2, \ldots, K$, if it is uncertain that object O_i has or does not have some attributes, U_i should be large.

Since the third property of U_i is not presented in a strict sense, various functions can be defined to satisfy these properties. For instance, let us assume that $K = 1$. In this case, only one attribute is concerned. The well-known *entropy* is a good uncertainty measurement:

$$U_i = \Psi(P_{i1}) = E(P_{i1}) = -P_{i1}\log P_{i1} - (1-P_{i1})\log(1-P_{i1}). \tag{2}$$

where E represents the entropy function. We will define uncertainty measurement for multiple attributes in Section III-C.

There is another important factor that affects the benefit the annotator can give to the system. It is the distribution of the objects in the low-level feature space. Suppose we have two objects that have the same uncertainty: one is at a high probability region in the low-level feature space where many other objects' feature vectors lie, and the other is at a very low probability region. Annotating these two objects will definitely give the system different amounts of information, which in turn leads to different retrieval performance. Therefore, we define the *knowledge gain* the annotator can give to the system by annotating object O_i as:

$$G_i = q_i \cdot U_i = q_i \cdot \Psi(P_{i1}, P_{i2}, \ldots, P_{iK}), \qquad i = 1, 2, \ldots, N. \tag{3}$$

where G_i is the defined *knowledge gain*; q_i is the *probability density function* around object O_i, which will be estimated in section III-A; U_i is the uncertainty measurement defined in (1). The criterion of choosing the next sample object is to find the unlabeled object O_i that has the maximum knowledge gain G_i.

III. The Proposed Approach

The proposed approach has a working flow as follows. We first initialize the probability lists with prior probabilities that we have about the whole database. The probability density function is also estimated. A small number of objects are randomly chosen and annotated as the initialization step of the algorithm. The probability list is re-calculated based on the randomly annotated objects. The system then start to select the object that has the maximum knowledge gain, and ask the annotator to annotate it. Again, some of the objects in the database update their probability lists because one of their neighbors is newly annotated. The system then searches for the object that has the maximum knowledge gain again, and the annotator is asked to annotate it. This loop keeps going until the annotator stops or the database is fully annotated.

In this section, we present the details of our proposed approach, including how to estimate the probability density function, how to update the probability list for each model, and how to calculate the expected knowledge gain if an object is annotated. These three problems are discussed in Section III-A, III-B and III-C, respectively.

A. Estimate the Probability Density Function

The probability density function is one of the important factors in the defined *knowledge gain* in Equation (3). This function can be calculated offline before annotation. In the machine learning literature, there have been many efficient ways for density estimation, such as the Naïve density estimator, the Bayesian networks, the mixture models, the density trees [34][35][36] and the kernel density estimator [33] (also known as Parzen windows). All of them can work well in some circumstances. In a normal content-based information retrieval setting, the number of objects in the database is typically large, which eases the density estimation. Most of the above algorithm can have fairly good results as long as the corresponding assumptions are correct. In the current system, we use the kernel density estimator simply because the kernel method is also used in updating the probability lists in the next subsection. Nevertheless, since the probability density estimation is independent to the latter probability list updating and needs only to be calculated once offline before the annotation, any of the above algorithms can be employed.

We choose the kernel as an isotropic Gaussian function (assume the features has been normalized), as it is widely used. The window of the estimation is a hyper-sphere centered at the concerned object O_i. Let the radius of the super-sphere be r_i, which was named the *bandwidth* of the kernel density estimator in the literature. Normally, $r_i = r, \text{for } i = 1,2,\cdots,N$, where r is a constant bandwidth. Let $\mathbf{x}_i$ be the feature vector of object O_i. The density estimation at the position where O_i locates is given by:

$$q_i = c\sum_{j=1}^{N} \text{kernel}(\mathbf{x}_i, \mathbf{x}_j) = c\sum_{j=1}^{N} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2r_j^2} \right) \qquad \text{for } i = 1, 2, \ldots, N. \tag{4}$$

where $\|\mathbf{x}_i - \mathbf{x}_j\|_2$ is the Euclidian distance from the neighboring object O_j to the center object O_i; c is a constant which does not matter when we compare the *knowledge gain*.

The choice of the bandwidth r has an important effect on the estimated probabilities. If the size of the neighborhood is too large, the estimation will suffer from low resolution. On the other hand, a too small size may cause local overfitting, which hurts the generalization ability of the estimation. The optimal Parzen window size has been studied extensively in the literature. The optimal bandwidth can be determined by minimizing the *Integrated Squared Error* (ISE) or the *Mean Integrated Squared Error* (MISE) [37][38]. Nevertheless, most of the work so far can only deal with low dimensional data. In a retrieval system, the dimension of the low-level features can easily pass tens or hundreds. Adaptive bandwidth was proposed in the literature to make the kernel density estimator works better in high dimensional space. In [39] and [40], Abramson suggested a square root law using $r_i \propto q_i^{-1/2}$, where r_i is the bandwidth of sample point $\mathbf{x}_i$. This proposal has been used in practice as a global estimator with surprisingly good small sample results. Nevertheless, in [41] Terrell and Scott performed a simulation study of the large sample properties of the Abramson estimator and notice some contrary results. For simplicity, we choose the bandwidth based on the maximum distance from any object to its closest neighbor. Through experiments we find that with a well-normalized feature space, a bandwidth of one to ten times that maximum distance often gives good results. Detailed experiments will be shown in Section V. We also examine the adaptive window method through experiments in Section V.

B. Update the Probability Lists

We assume that we have some very rough knowledge about the probability lists before the annotation. That is,

$$P_{ik} = P^{(k)}, \qquad \text{for } i = 1, 2, \ldots, N, k = 1, 2, \ldots, K. \tag{5}$$

where $P^{(k)}$ is the prior probability for an object to have attribute A_k. Experimental results show that the guess of the prior probability will not influence the annotation efficiency too much. During the annotation, the annotator is supposed to check all the attributes the query model has, and all the other attributes the annotator does not check are assumed to be not belonging to the object unless they have some of their children nodes checked. Let Θ_i be the set of attributes the annotator annotated for object O_i, including those having children nodes checked. The new list of probabilities for object O_i after the annotation is:

$$P_{ik} = \begin{cases} 1, & \text{if } A_k \in \Theta_i, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Recall the basic assumption we made in Section I, annotated models tend to infer knowledge to their close by neighbors. If a model has some of its neighbors annotated, its probability list needs to be updated. Meanwhile, if the objects are far from all the annotated object, we do not want to link the semantic meanings between them. This favor of semantic meaning extension fits the framework of kernel regression very well. The annotated objects are anchor points that have known probability values. For those objects that haven't been annotated, their probabilities of having some attributes can be regressively interpolated. Although for each object in the database, there are a list of attributes associated with it, we assume in this paper that for each attribute the probabilities can be independently interpolated with kernel regression.

Kernel regression is essentially a weighted average method to do interpolation. Let $\mathbf{a}_m, m=1,\cdots,M$ be all the anchor points for which we know the values $u_m, m=1,\cdots,M$. Given a new point $\mathbf{x}$ that we want to find its value u, kernel regression gives:

$$u = \frac{\sum_{m=1}^{M} w_m u_m}{\sum_{m=1}^{M} w_m} \tag{7}$$

If we still use Gaussian function as our kernel, the weights are defined as:

$$w_m = \mathrm{kernel}(\mathbf{x}, \mathbf{a}_m) = \exp\left(-\frac{\left\| \mathbf{x} - \mathbf{a}_m \right\|_2^2}{2 r_m'^2} \right), m = 1, \cdots M. \tag{8}$$

where r_m' is the bandwidth used for anchor point $\mathbf{a}_m$ in the kernel regression. This bandwidth will have very similar effects on the final result as that when we estimate the probability density function using kernel density estimator. Actually, we will use the same kernel bandwidth in the probability density function estimation in the last subsection and kernel regression here. Obviously, from Equation (8), an anchor point that is closer to the query point $\mathbf{x}$ will be assigned a higher weight, which gives more influence on the predicted value u. This is coherent with our basic assumption.

As we mentioned before, if an object O_i is annotated as having attribute A_k, the probability P_{ik} will increase to 1. Otherwise, it will drop to 0. These annotated objects are considered as anchor points in the low-level feature space. Let us consider for example one of the attributes A_k. Let $\tilde{\mathbf{x}}_m, m = 1,\cdots, M$ be the feature vectors of all the currently annotated objects, and the corresponding probabilities P_{mk} are defined as in Equation (6), i.e.,

$$P_{mk} = \begin{cases} 1, & \text{if the object corresponding to } \tilde{\mathbf{x}}_\mathrm{m} \text{ has } A_k, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Follow the formulization of kernel regression, given an un-annotated object whose feature vector is $\mathbf{x}$, the probability of this object having attribute A_k is:

$$P(\mathbf{x} \in A_k) = \frac{\sum_{m=1}^{M} w_m P_{mk}}{\sum_{m=1}^{M} w_m} \tag{10}$$

where the weights are:

$$w_m = \exp\left(-\frac{\|\mathbf{x}-\tilde{\mathbf{x}}_m\|_2^2}{2r_m'^2}\right), m = 1,\cdots M. \tag{11}$$

Again, r_m' is the bandwidth for object $\tilde{\mathbf{x}}_m$.

There is a problem with the above regression algorithm. A simple 1D example is shown in Figure 2 (a). The horizontal axis is the feature value, and the vertical axis is the probability that the corresponding object has the certain attribute. Notice that although the feature value is far away from the anchor points (e.g., at the two ends of the horizontal axis), and the weight is very small, the predicted probability is still close to 1 or 0. This is mainly due to the normalization of the weights at the denominator of Equation (10). However, this effect is not what we expected. Again, our assumption is that close annotated objects can infer knowledge to the current object, but far objects should not. In other words, if an object has only very far neighbors being annotated, we expect its probability to remain the prior probability.

To solve this problem, we propose a simple algorithm called *biased kernel regression*. Basically, we modify the Equation (7) as:

$$u = \frac{\sum_{m=1}^{M} w_m u_m + w_0 u_0}{\sum_{m=1}^{M} w_m + w_0} \tag{12}$$

where u_0 is the *bias value* of the point to be predicted; w_0 is a weight representing the strength of the bias. A large weight will produce a prediction value very close to the bias value. If the prior knowledge has a high confidence, the weight should be set large. If we have enough prior knowledge, the weight can be adaptive in the low-level feature space. When the weight w_0 is less than 1, there exists an equivalent distance r_0, which satisfies:

$$w_0 = \exp\left(-\frac{r_0^2}{2r'^2}\right) \tag{13}$$

where r' is the kernel bandwidth at the to be predicted object. In this case, biased kernel regression can be viewed by putting a virtual anchor point at a distance r_0 to the point to be predicted, and set the value of the virtual anchor point as the bias value.

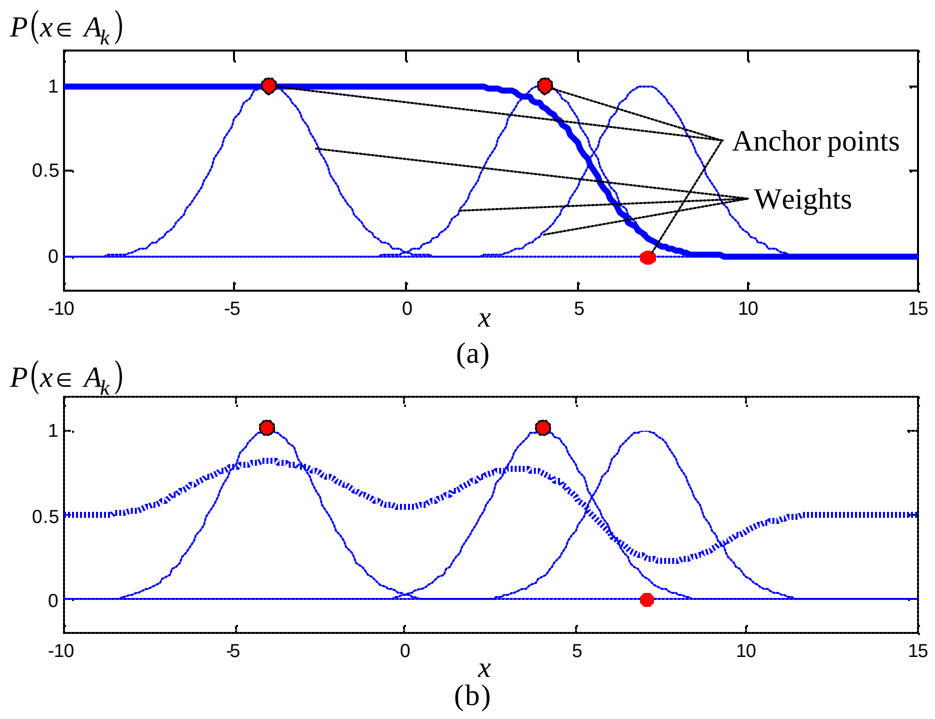


Figure 2 The normal kernel regression (a) and the biased kernel regression (b).

In our system, the bias value during the probability estimation is the prior probability $P^{(k)}$. Since the prior probability is a rough estimate, which is uniform across the low-level feature space, the weight w_0 is preferred to be small. In Figure 2 (b), we show the result using biased kernel regression on the same setting of Figure 2 (a). r_0 is set to be equal to r', and the prior probability is set to be 0.5. Notice that when the predicted points are far from the anchor points, the predicted value goes to the prior probability. This is what we expected. Also notice that the predicted curve does not pass the anchor point, for both figures in Figure 2. This is actually a nice property of kernel regression. From the probabilistic point of view, if the object with a certain feature vector is annotated as having attribute A_k, there might be still non-zero probability that another object having the same feature value does not have this attribute. In this sense, what we called "probability" for the annotated objects, which has the value 1 or 0, is not the real probability at the corresponding point in the low-level feature space. They are just some values that can help us estimate the probability list of the neighboring un-annotated objects. Nevertheless, it can be proved that when the number of anchor points goes infinite and the kernel bandwidth becomes very small, the result of kernel regression asymptotically converges to the actual probability distribution [31].

Although the database might be huge, the computational cost on the probability list updating is actually low. That's because in kernel regression, a newly annotated object would not change an object's probability list if it were far away. The bandwidth of the kernel function determines the hyper-sphere inside which the object's probability lists have to be updated. These objects can be easily found if the database is organized by R-tree or similar structures.

Updating of the probabilities can also be done under other frameworks. We choose kernel regression because it provides us a natural way to deal with the problem and a smooth transition from no annotation to full annotation. Once the database is fully annotated, the probability lists are equivalent to full annotation. The probabilities of the objects can also be estimated parametrically. For example, we can assume the models belonging to each attribute follow a Gaussian distribution, or a Gaussian mixture distribution if necessary. Having a new model annotated is equivalent to adding a new training example to the Gaussian or the Gaussian mixture. The model to be chosen to annotate could be the one with which the system is the least confident. Such an approach does not have a smooth transition from no annotation to full annotation, because after the database is fully annotated, the parametric model does not record any annotation for each object in the database. Moreover, such kind of approach imposes a very strong global structure over the low-level feature space. When the model of the distribution is not right, the performance of the system may suffer a lot. As a comparison, our approach also imposes some structures on the feature space but they are local. Local structure offers better opportunity to fit the database well when we do not have enough knowledge about the database.

C. The Uncertainty Measure

After all the probabilities have been updated, the learning algorithm searches among models that have not been annotated for another model whose annotation, once given by the annotator, will provide the most extra information. According to the discussion in section II-B, this model is the one that produces the maximum knowledge gain. In order to calculate the gain, we need to find the uncertainty measurement and the probability density for each model. We have described the estimation of the probability density function in Section III-A. Hereinafter we discuss the way to determine the uncertainty measurement.

In section II-B, we gave some general properties for the uncertainty measurement we want. We mentioned that if we only have one attribute to annotate for all the objects, the *entropy* is a good measure of uncertainty

$$U_i = \Psi\left(P_{i1}\right) = E\left(P_{i1}\right) = -P_{i1}\log P_{i1} - (1-P_{i1})\log(1-P_{i1}) \tag{14}$$

where U_i is the uncertainty measurement for object O_i, E is the entropy function, P_{i1} is the probability for object O_i to be characterized by the attribute. In the real case, we have multiple attributes in the database. The uncertainty should be defined based on the joint probability of all the attributes. That is:

$$\begin{aligned} U_i &= \Psi\left(P_i\left(A_1,\cdots,A_K\right)\right) \\ &= E\left(P_i\left(A_1,\cdots,A_K\right)\right) \\ &= -\sum P_i\left(A_1,\cdots,A_K\right)\log P_i\left(A_1,\cdots,A_K\right) \end{aligned} \tag{15}$$

where $P_i\left(A_1,\cdots,A_K\right)$ represents the joint probability of object O_i having or not having the attributes. The sum is taken over all the possible combinations of attributes that an object can have.

There are two problems if we want to find the probability of all the combinations and then calculate the entropy as the uncertainty measure. We will use a toy attribute tree as in Figure 3 to illustrate the two problems.

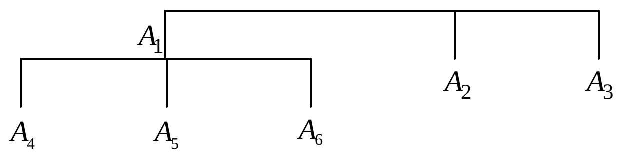


Figure 3 A toy attribute tree.

The first problem is that the joint probability is not available. Instead, we know the probability for each attribute separately. For example, in Figure 3 we know $P(A_1), P(A_2), \cdots, P(A_6)$, which are estimated in the probability list updating stage. We can also make some reasonable assumptions. As we mentioned in Section II-A, in this paper, we assume that the attributes at the same level of the tree structure are not exclusive. Therefore, during the estimate of the joint probability, we can assume them to be independent. Even the attributes at the same level may not be actually independent, we are still confident in the sense that at least we are estimating the upper bound of the uncertainty. Including the tree structure assumption, for Figure 3 we have:

$$\begin{cases} P\left(A_1, A_2, \cdots, A_6\right) = P\left(A_1, A_4, A_5, A_6\right) \cdot P\left(A_2\right) \cdot P\left(A_3\right) \\ P\left(A_4, A_5, A_6 \middle| A_1\right) = P\left(A_4 \middle| A_1\right) \cdot P\left(A_5 \middle| A_1\right) \cdot P\left(A_6 \middle| A_1\right) \\ P\left(\mathbf{x} \in A_1 \middle| \mathbf{x} \in A_4\right) = P\left(\mathbf{x} \in A_1 \middle| \mathbf{x} \in A_5\right) = P\left(\mathbf{x} \in A_1 \middle| \mathbf{x} \in A_6\right) = 1 \end{cases} \tag{16}$$

However, the above assumptions are not enough yet for finding the joint probability of any combination of attributes. Further assumptions such as $P\left(\mathbf{x} \in A_1 \middle| \mathbf{x} \notin A_i\right), i = 4, 5, 6$ have to be made, which is difficult to justify.

The second problem is more serious. In order to find the uncertainty in Equation (15), even for attribute tree as simple as Figure 3, we have to calculate 2^6 items. This number increases exponentially with the number of attributes associated with the database. This prohibits the algorithm being used for large and complex databases.

We take a rather simplified way to find the uncertainty measure. Still use the toy attribute tree in Figure 3 as an example. With all the conditions given in Equation (16), we have:

$$\begin{aligned} U &= \Psi(A_1, A_2, \cdots, A_6) \\ &= E(A_1, A_2, \cdots, A_6) \\ &= E(A_1, A_4, A_5, A_6) + E(A_2) + E(A_3) \\ &= E(A_4, A_5, A_6 | A_1) + E(A_1) + E(A_2) + E(A_3) \\ &= E(A_4|A_1) + E(A_5|A_1) + E(A_6|A_1) + E(A_1) + E(A_2) + E(A_3) \end{aligned} \tag{17}$$

where E is the entropy function, and $E(a \mid b)$ is the conditional entropy of a given b. As conditional entropy is always smaller than or equal to the un-conditional entropy, we have:

$$E(A_4|A_1) \le E(A_4); E(A_5|A_1) \le E(A_5); E(A_6|A_1) \le E(A_6) \tag{18}$$

Notice that A_1, A_4, A_5 and A_6 are in a tree structure; the inequalities above are actually strict. Computing the conditional entropy requires making further assumptions and more computations. To reduce the complexity during the learning process, we simplify it as follows:

$$E(A_4|A_1) = aE(A_4); E(A_5|A_1) = aE(A_5); E(A_6|A_1) = aE(A_6) \tag{19}$$

where a is a constant between 0 and 1. The overall uncertainty becomes:

$$\begin{aligned} U &= \Psi(A_1, A_2, \cdots, A_6) \\ &= E(A_1) + E(A_2) + E(A_3) + a\left[E(A_4) + E(A_5) + E(A_6)\right] \end{aligned} \tag{20}$$

which is a weighted sum of the individual entropies for all the attributes.

In general, for a certain object O_i and a certain attribute A_k, we define the individual entropy as:

$$E_{ik} = -P_{ik} \log P_{ik} - (1 - P_{ik}) \log(1 - P_{ik}). \tag{21}$$

The overall uncertainty for an object O_i is defined by a weighted sum of the entropies for all the attributes, i.e.,

$$U_i = \sum_{k=1}^{K} w_{Sk} E_{ik} \tag{22}$$

where K is the total number of attributes, and w_{Sk} is the semantic weight for each attribute. The semantic weights are related with which level in the tree the attributes are at. Let ℓ_k be the level attribute A_k is at. For example, in the tree structure in Figure 3, attributes such as A_1, A_2 and A_3

are at the first level ($\ell_k = 1$), and A_4, A_5 and A_6 are at the second level ($\ell_k = 2$). The weights are defined as:

$$w_{Sk} = a^{\ell_k - 1} \tag{23}$$

where a is a constant between 0 and 1. In our current implementation, we set a to be 0.6 based on experiments.

With the uncertainty measure in Equation (22) and the probability density estimate in Section III-A, we are able to calculate the knowledge gain by simply multiplying them together as in Equation (3). The system then proposes the object with the maximum gain and asks the annotator to annotate it. After the annotation, the system updates the probability lists, recalculates the uncertainty measures and proposes the next sample. This loop keeps going until the annotator stops or the database is fully annotated.

IV. Joint Similarity Measure for Semantic and Low-Level Features

The hidden annotation needs to be integrated into the retrieval system in order to provide better retrieval performance. In previous work, annotation was often regarded as a Boolean vector. In [11] and [12], normalized Hamming distance was used to combine the influence of the annotation and acted as a new feature for the retrieval. When the database is partially annotated and the annotations are used for learning, as in [14], neural networks are often used to train the similarity measurement.

In our system, each model has a list of probabilities of having the attributes, including the query model the user provides. If the query model is chosen from the database, we already have this probability list. This is the normal case as hidden annotation is largely for improving the performance of inside-database queries. If the query model is selected from outside the database, we can estimate its probabilities as in Section III-B as well. Alternatively, the user can annotate the query model before providing it to the retrieval system. The probability list is a complete description of all the annotations we have ever made and is associated with high-level semantics. We can treat this list of probabilities as a feature vector, similar to low-level features such as color, texture, and shape. The semantic distance d_{S12} between any two objects O_1 and O_2 is defined as:

$$d_{S12} = \sum_{k=1}^{K} w_{Sk} \left[P_{1k}(1 - P_{2k}) + P_{2k}(1 - P_{1k}) \right] \tag{24}$$

where K is the total number of attributes, w_{Sk} is the semantic weight for each attribute as defined in (23), and P_{1k} and P_{2k} are the attribute probabilities for the two models. The item

$P_{1k}(1-P_{2k})+P_{2k}(1-P_{1k})$ is actually the probability of objects O_1 and O_2 disagree with each other on attribute A_k (i.e., one of them has A_k but the other one does not have). We choose the form of weighted sum to measure the overall disagreement because it's simple and effective in practice. For attributes at a higher level, the weight is smaller, so that we give a less penalty on disagreement on high level attributes. Intuitively, the disagreement between a "car" and an "aircraft" is larger than that between "Classic Aircraft" and a "Jets Aircraft". Another good property of the defined semantic distance is that, if the to be compared objects have been annotated and the probabilities are either 0 or 1, the defined semantic distance will automatically degenerate into a Hamming distance (assume one-level attribute tree), which is widely used in the literature. Although other form of semantic distance can be defined, notice that it's not reasonable to normalize the probability lists and measure their difference using KL divergence. As we have stated, the attributes in our system are not exclusive, and viewing them as one probability distribution is not logical.

We need another distance measure that is the distance in the low-level feature space. For two objects O_1 and O_2, we simply use the weighted Euclidean distance

$$d_{L12} = \sqrt{\sum_{j=1}^{J} w_{Lj}(f_{1j} - f_{2j})^2} \tag{25}$$

where J is the total number of features, f_{1j} and f_{2j} are the j^{th} normalized low-level features of the two objects O_1 and O_2, and w_{Lj} is the weight set based on the importance of each feature. In the current implementation, the features are equally weighted after normalization.

The overall distance between the two models is a weighted sum of the semantic distance and the low-level feature distance:

$$d_{Overall12} = w_S \cdot d_{S12} + w_L \cdot d_{L12} \tag{26}$$

where w_S and w_L are the semantic weight and the low-level feature weight respectively and $w_S + w_L = 1$. There are several methods to specify these two weights. For example, they can be fixed as a constant, or they can be proportional to the number of objects that have been annotated in the whole database. In the current system, the weights are determined by the query object:

$$w_S = 2w_{S\max} \cdot \left(\max_{k=1,\cdots,K}(P_{qk}) - 0.5 \right) \tag{27}$$

where P_{qk} is the probability for the query object to have attribute A_k, and $w_{S\max}$ is a constant which is currently set to be 0.9. If the query object has been annotated, or its estimated probabilities have maximum value 1, the weight of the semantic distance is the maximum, i.e.,

$w_{S\max}$. In other words, if the retrieval system knows that the query has a certain attribute, it is preferable to search the database mainly by this attribute instead of the low-level features. If the maximum of the probabilities is less than 0.5, the semantic weight will be set as 0.

As hidden annotations are made by the annotator, models that are far away from each other may be annotated as having the same attributes, which means they have small semantic distance. The integration of semantic annotations into the similarity measurement effectively works as warping of the low-level feature space to make semantically similar objects closer to each other. As illustrated in Figure 4, aircrafts are distributed beside cars in the low-level feature space. Two aircrafts and one car are annotated. When we compute the final similarity, the aircrafts will have similar attribute probabilities, and their final similarity score will be higher than when only low-level features are considered due to the introducing of item $w_S \cdot d_{S12}$ in (26).

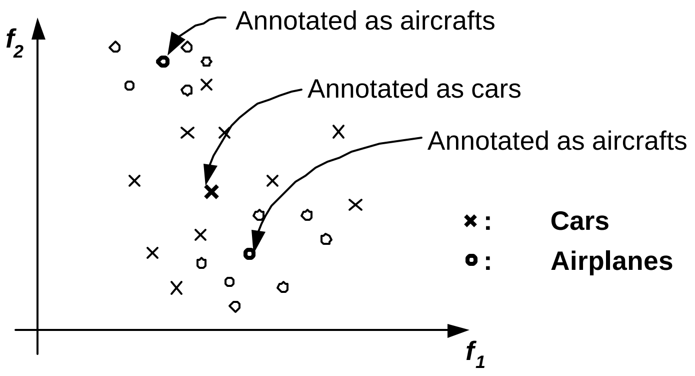


Figure 4 The annotations will "warp" the feature space.

V. Experiments

We first test our algorithm on a synthetic database. There are 2000 objects in the database, which fall into 3 categories and 2 subcategories. The attribute tree of the synthetic database is shown in Figure 5. The dimension of the low-level feature space is two. The distribution of the categories in the feature space is shown in Figure 6.

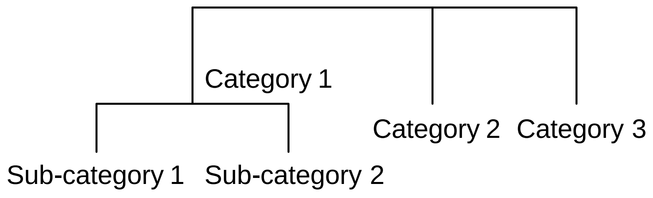


Figure 5 The attribute tree of the synthetic database.

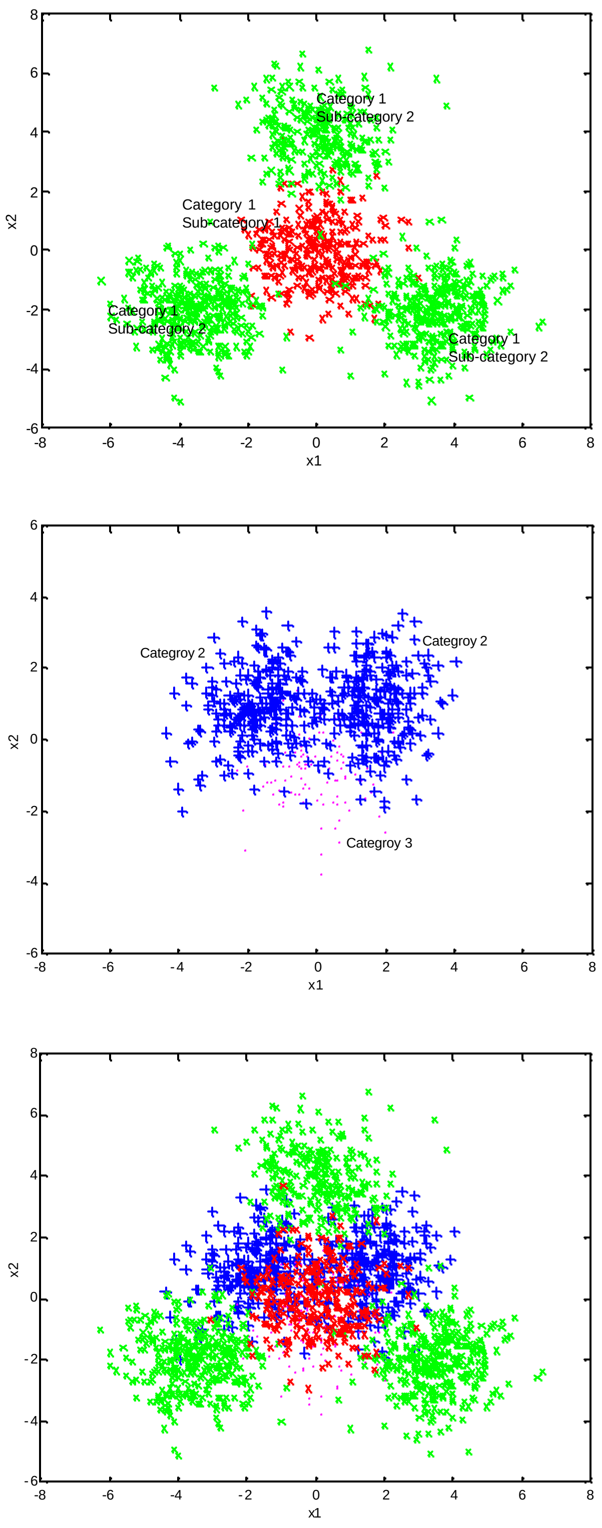


Figure 6 The synthetic database

From Figure 6 we can see that the features of Category 2, Category 3 and Category 1 Subcategory 1 overlap a lot in the low-level feature space. This will hurt the performance of the retrieval system. We want to do hidden annotation to improve the performance.

We measure the annotation efficiency by testing the final retrieval performance of our retrieval system. Since our system has a multi-level attribute tree structure, we define our own performance measurement as follows. For any specific query q and its top R retrieved results, the average matching error for these results is measured by:

$$e_S^q = \frac{1}{R} \sum_{\substack{j \in top\ R\ results \\ of\ query\ q}} d_S^{qj} \tag{28}$$

where d_S^{qj} is the semantic distance between the query and the j^{th} retrieved object, which is calculated by (24) with the ground truth data. The e_S^q indicates the average matching error for the top R retrieved objects with respect to the query. The smaller the e_S^q, the better the performance of the system for the query q. The overall system performance is evaluated by:

$$Err = \frac{1}{N} \sum_{i=1}^{N} e_s^i \tag{29}$$

where N is the number of objects in the database, as we take every object in the database as a query and calculate the average matching error. The final performance of the system is measured by taking average of the average matching error for all the objects.

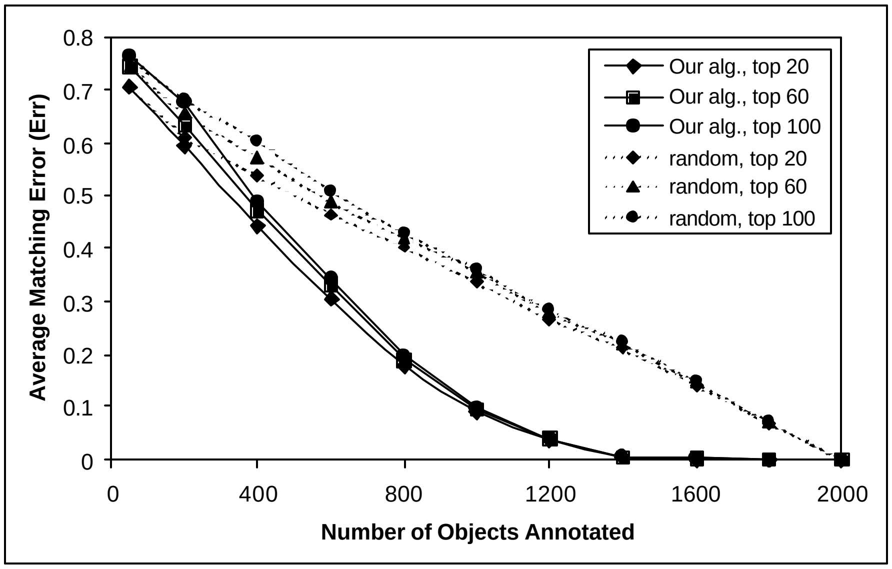


Figure 7 Performance comparison between our algorithm and random sampling.

Figure 7 shows the performance comparison between our active learning algorithm and the random sampling algorithm. In our algorithm, we use fixed bandwidth for kernel density estimation and kernel regression. The bandwidth is set to be twice the maximum distance from any object to its closest neighboring object. Later we will show experiment on how to choose the bandwidth. In the biased kernel regression, we choose the weight of the prior probability as e^{-2}. We will also discuss the weight selection later. The first 50 annotated objects are randomly drawn. After that, our algorithm use active learning to pick the samples to annotate while the random sampling algorithm keeps sampling randomly.

The horizontal axis of Figure 7 is the number of samples that have been annotated. The vertical axis is the average matching error for the whole database measured by (29). A curve closer to the bottom-left corner is considered to have a better performance. Six curves are shown in the figure, representing the performance of different algorithms measured by different number of retrieved results. The average matching errors of the top 20, top 60 and top 100 retrieved results are reported. As we expected, every curve drops as the number of annotated samples increases, which shows the effectiveness of hidden annotation. Furthermore, the dropping slope of our algorithm is much steeper than that of the random sampling algorithm, which shows that the active learning algorithm performs better. With active learning, given a certain average match error target, the save of the number of annotations can be as large as 50% compared with random sampling. Notice that both algorithms have zero average matching error when the number of annotated object is close to 2000. This is because the database is about to be fully annotated, and the performance of any algorithm will have zero average matching error at this point.

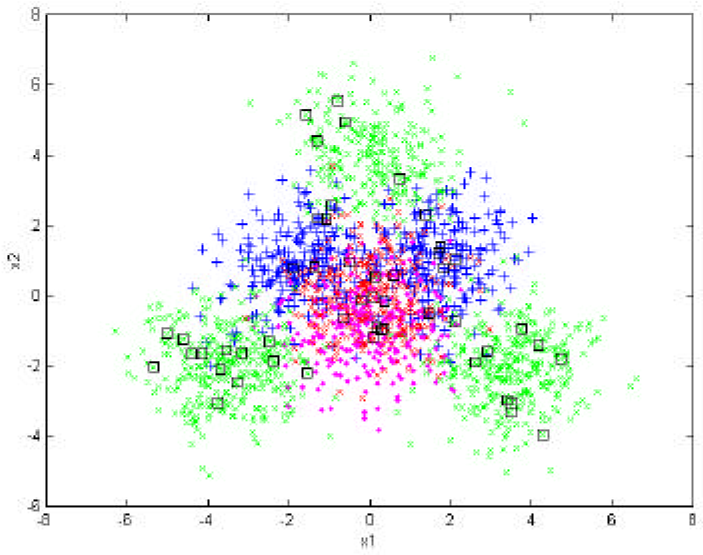
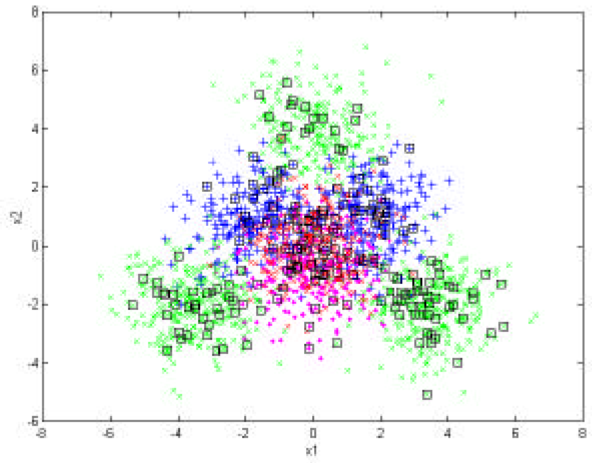
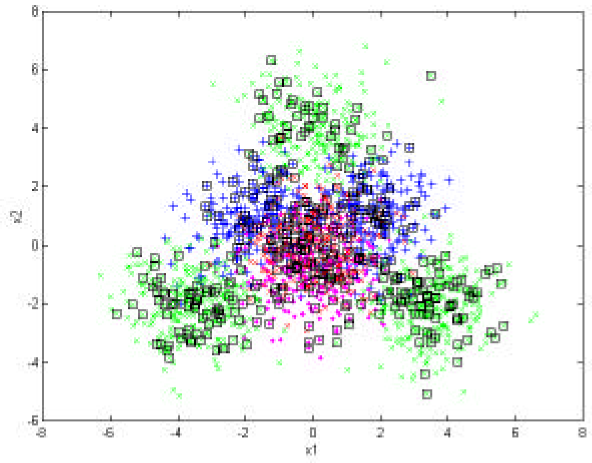


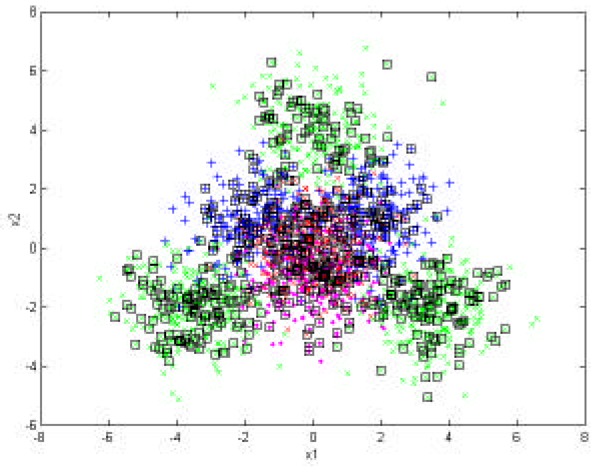
Figure 8 In the initialization step, 50 objects are chosen randomly and annotated.



(a)

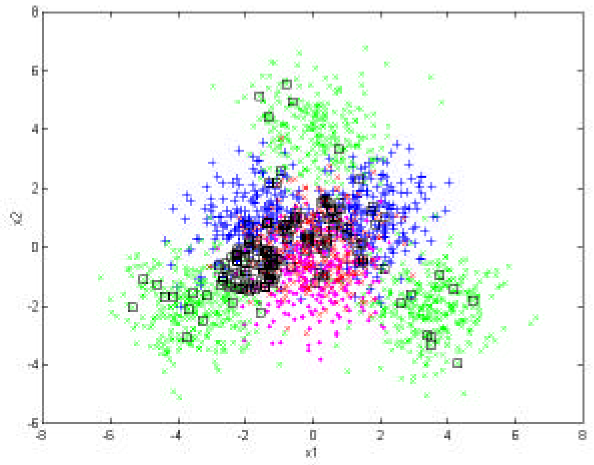


(b)

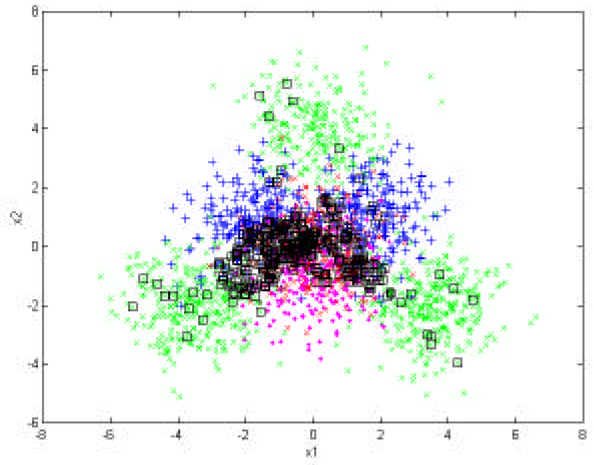


(c)

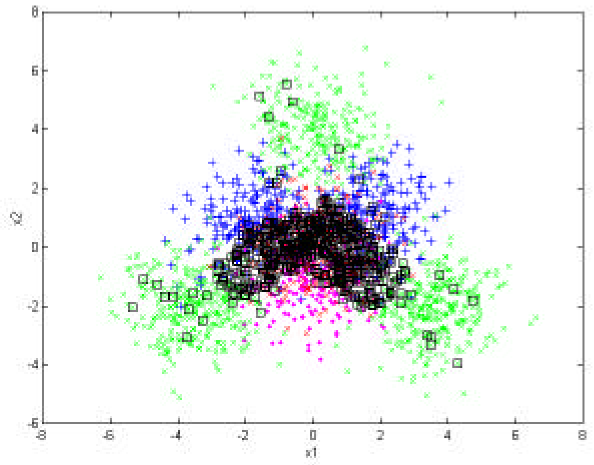
Figure 9 The annotation process of random sampling.
(a) 200 objects annotated. (b) 400 objects annotated. (c) 600 objects annotated.



(a)



(b)



(c)

Figure 10 The annotation process of our active learning algorithm.

(a) 200 objects annotated. (b) 400 objects annotated. (c) 600 objects annotated.

To better understand what happens during the annotation process, we plot the annotated objects in Figure 8, Figure 9 and Figure 10. In these figures, we use black squares to represent an annotated object. From Figure 9 we can see that random sampling wastes a lot of annotation on areas that the low-level feature is already very good for retrieval. As a comparison, in Figure 10, the active learning algorithm focuses on annotating the confusing area and leaves the unconfusing area untouched. This is the major reason that active learning can outperform the random sampling algorithm.

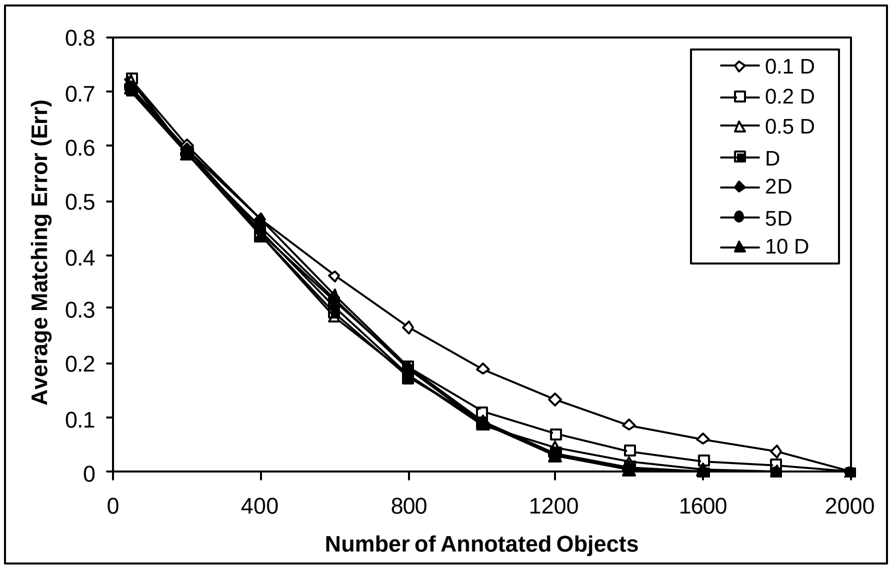


Figure 11 Select the right kernel bandwidth.

Next we explore the sensitivity of the kernel bandwidth selection to our algorithm. Fixed kernel bandwidth is employed throughout the experiment. We find that the maximum distance from any object to its closest neighboring object is a good choice if the features are well normalized. Let this distance be *D*. Figure 11 shows the retrieval performance at different kernel bandwidths, i.e., 0.1*D*, 0.2*D*, 0.5*D*, *D*, 2*D*, 5*D*, 10*D*. Only the performance of the top 20 retrieved results is reported, and the other settings are the same as in the last experiment. It can be observed that for a fairly large range around *D*, the performance is stable. In addition, choosing a relatively large bandwidth is less risky than choosing a small bandwidth.

An alternative of fixed kernel bandwidth (FKW) is the adaptive kernel bandwidth (AKW). We try a simple adaptive kernel bandwidth algorithm described in [33] and the results are shown in Figure 12. The algorithm first estimates the probability density function with a fixed kernel bandwidth, and then adjusts the kernel bandwidth at each sample point based on the estimated probability density function using the square-root law [39][40]. Notice that even in the adaptive kernel density estimation, we still need to choose a base bandwidth, which will be tuned by a local bandwidth multiplication factor generated by the square-root law. In Figure 12 the bandwidth of the adaptive kernel bandwidth algorithm is the base bandwidth we choose. Although adaptive kernel bandwidth is for probability density function estimation, the same bandwidth is used to interpolate the probabilities during the kernel regression stage.

From Figure 12 we can see that for a good kernel bandwidth selection, e.g., 2*D*, adaptive kernel bandwidth performs almost the same as the fixed kernel bandwidth. While for a bad kernel bandwidth, e.g., 0.1*D*, adaptive kernel bandwidth actually performs worse. Though the reason for this is not justified yet, we use fixed kernel bandwidth in the left experiments.

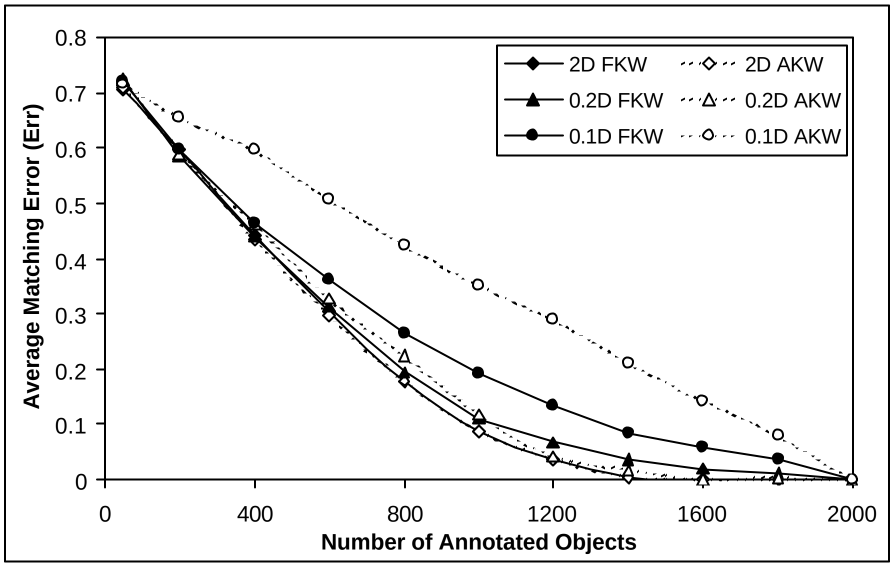


Figure 12 Fixed kernel bandwidth vs. adaptive kernel bandwidth.

Finally we want to see the effect of choosing different weights for the prior probability in the biased kernel regression (Equation (12)). The results are shown in Figure 13. All the other settings are the same as the first experiment in this section. Surprisingly, weight does not change the performance of the system too much. Although not significant, two interesting phenomena can be observed from the figure. First, there is a small divergence of the performance when very few objects are annotated (e.g., when only 50 models are annotated) and a convergence of the performance when more and more objects are annotated. This is because when more and more objects are annotated, there will be always enough effective annotated neighboring objects found during the kernel regression, and the bias to the prior probability will be ignored. Second, there is a performance intercrossing at the very beginning of the curves. Namely, when very few objects in the database are annotated, a larger bias weight gives better performance because it is resistant to overfitting. When more and more objects are annotated, a smaller weight will give better performance because it is easier to quickly adapt to the local probability changes. This phenomenon suggests an adaptive weighting scheme across the annotation process, which can be our future work. Nevertheless, the difference of the performance is rather small and can be ignored in the current stage.

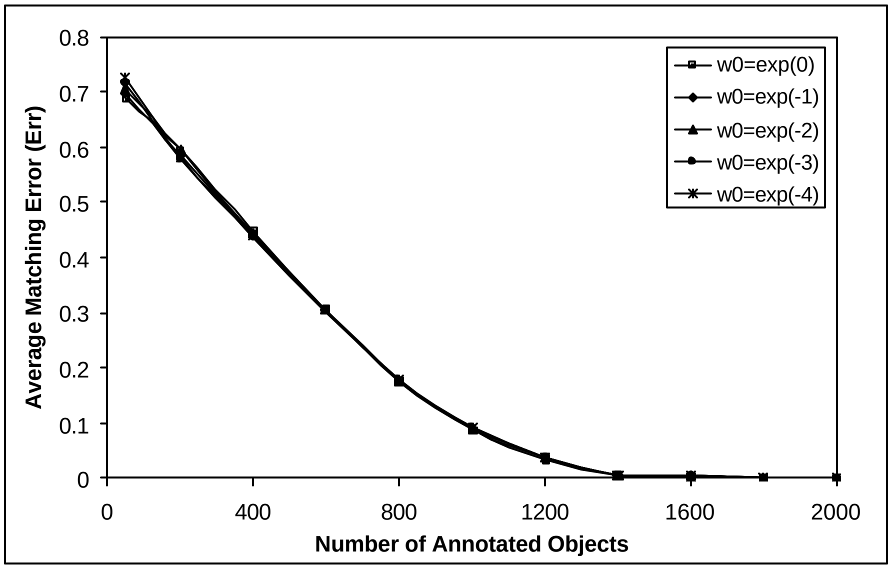


Figure 13 The selection of the weight for the prior probability in the biased kernel regression.

We then test our algorithm on a real retrieval system. This is a 3D model database in which most of the 3D models are downloaded from the Internet. The database consists of 1750 objects, whose categories are highly biased. More than one third of the objects are aircrafts. 3D model retrieval is a relatively new research area and not too much work has been published. Some features for comparing 3D models has been proposed in the literature, e.g., those in [16][17][18][19][20][21]. In our system, we have 10 features extracted for each object. They are region-based features proposed in [21], including the volume-surface ratio, the aspect ratio, moment invariants and Fourier transform coefficients. The features are normalized to be within range $(-1,\ 1)$.

The low-level features for 3D models proposed so far are far from satisfying. That's the reason why we want to integrate hidden annotation to improve the retrieval performance. In this experiment, we use our active learning algorithm to distinguish between aircrafts and non-aircrafts. Because some of the 3D models contain multiple objects, 100 of them are annotated as both aircraft and non-aircraft.

The performance comparison between our active learning algorithm and the random sampling algorithm on the 3D model database is given in Figure 14. To start the algorithms, 50 models are randomly chosen and annotated. The setting of our algorithm is the same as that of the synthetic database. That is, we use fixed bandwidth for kernel density estimation and kernel regression. The bandwidth is set to be twice the maximum distance from any object to its closest neighboring object. In the biased kernel regression, we choose the weight of the prior probability as e^{-2}. Only the performance on the top 20 retrieved results are reported. From Figure 14, it is obvious that our active learning algorithm works much better than the random sampling approach.

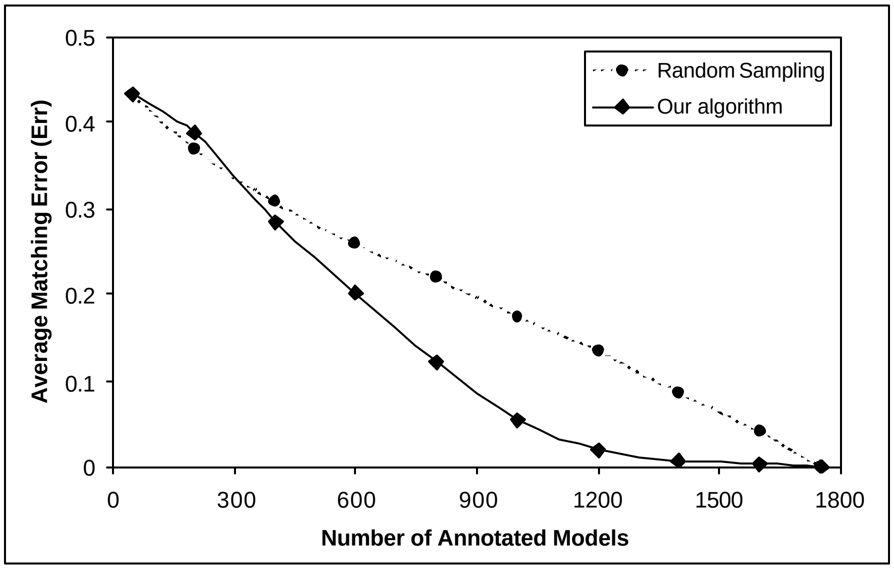


Figure 14 Performance comparison between our algorithm and random sampling for the 3D model database.

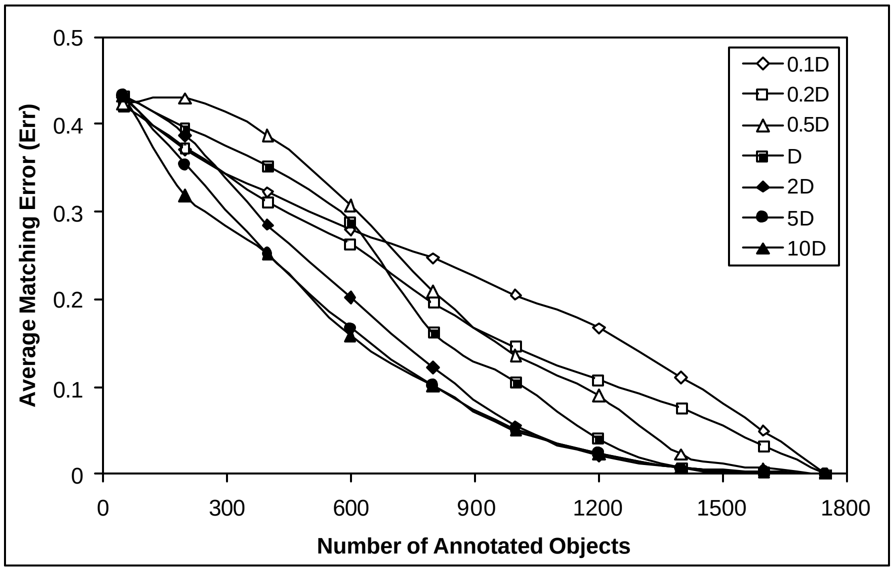


Figure 15 Select the kernel bandwidth on the real 3D model database.

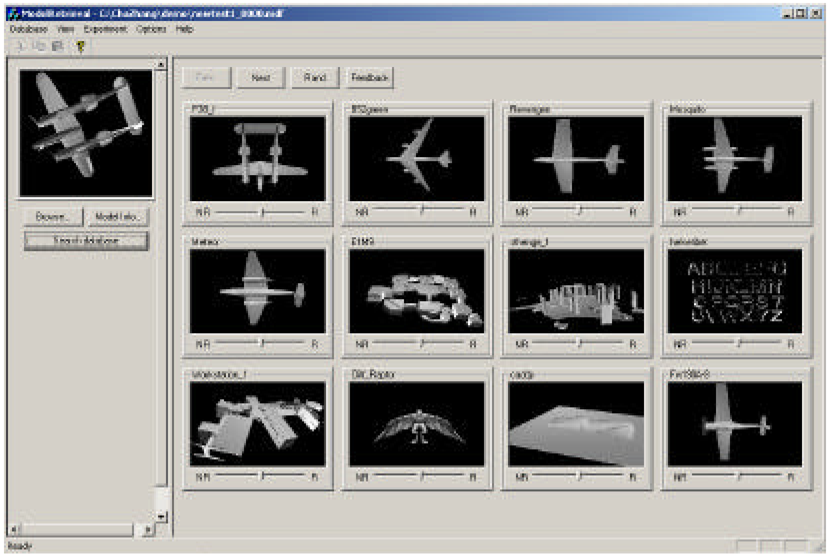
Since the choice of the kernel bandwidth is important for density estimation and kernel regression, we test the retrieval performance with respect to bandwidth changing again. The results are shown in Figure 15. We can see that although the variation of the performance is larger than that of the synthetic database (Figure 11), larger kernel bandwidth still provides better results in general. However the bandwidth cannot be infinite, because too large bandwidth will smooth out too much details in the feature space. Another thing to notice is that the larger the kernel bandwidth employed, the more computational cost we have to pay during the probability list updating. We need to have some tradeoff between performance and speed.

VI. Conclusions and Discussions

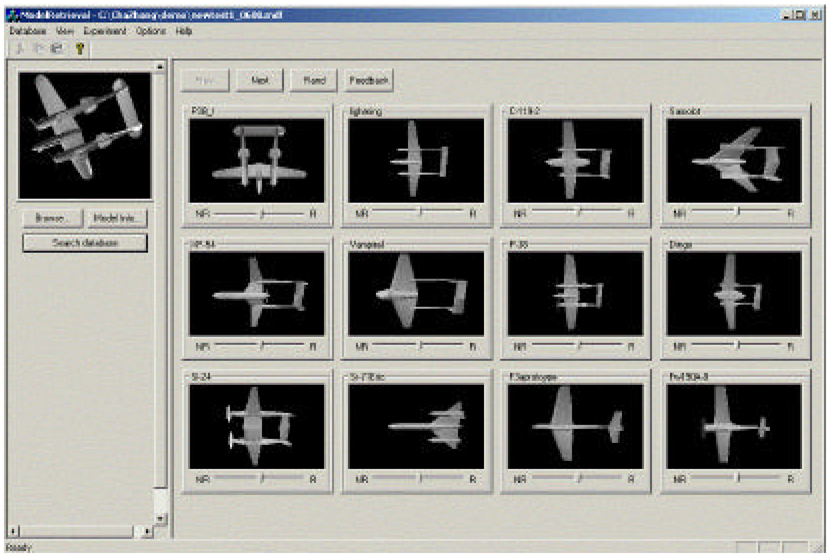
In this paper, we proposed a general approach to make hidden annotation with active learning for information retrieval. We considered a natural attribute tree structure for the annotation. The object to be annotated next was determined by the knowledge gain of the system by annotating it. We defined the knowledge gain as the product of the probability density function and the uncertainty measurement. In order to evaluate the uncertainty of an object, we gave each object a list of attribute probabilities, computed based on its neighboring annotated objects through kernel regression. We obtained the uncertainty of an object by giving an explicit function on these probabilities. The proposed algorithm outperforms the random sampling algorithm in all the experiments, which shows that hidden annotation with active learning is a very powerful tool to help improve the performance of content-based information retrieval.

The relationship between relevance feedback and hidden annotation is worth discussing. In Section I, we mentioned that we treat them as two separate stages of a retrieval system. This is because the user's measurement for similarity may change from time to time and from user to user. If every user's feedback is taken as annotation, some of them may even deteriorate the overall system performance. Hidden annotation through active learning has shown its effectiveness on improving the within-database query results. However, with only a few annotated models, there are typically not enough data to train the system well on every attribute. Over-fitting to the small amount of training data is possible. Experiments show that for an outsider query of the database, the improvement of the performance is not significant. Therefore, relevance feedback is still necessary in a retrieval system.

The relevance feedback and hidden annotation are two different learning strategies. Relevance feedback can adapt to the query very quickly and is designed to work specifically for each query. Hidden annotation with active learning is more like a student who tries to learn as much as possible and as fast as possible, and the result is an improvement for the overall performance. Whether or not to accumulate previously learned knowledge is one of the major differences between relevance feedback and hidden annotation. As a result, if the user who gives the feedback is trustworthy, this feedback can be accumulated and added to the knowledge of the system, as was done in [12]. Otherwise, we would better separate hidden annotation and relevance feedback completely and give the privilege of annotation only to the system designer or the annotator.



(a) retrieval results without annotation



(b) retrieval results with 1/3 models annotated

Figure 16 Retrieval results for the 3D model database with/without partial annotation.

One may have the concern whether the annotator's preference is always the same as a user. Although the annotator may be an expert, the user may have his/her own criteria for similarity. We can modify our algorithm to allow user feedback. Because we define the overall distance as a weighted sum of the semantic distance and the low-level feature distance as in (26), where the weights between semantic distance and low-level feature distance can also be adjusted by the relevance feedback. This is part of our future work.

To compare our approach with Lewis and Gale's approach in [25], their approach is designed for text classification, while ours is for information retrieval. A two-class problem is considered in their paper, while our approach deals with multiple attributes forming complex tree structure. They estimate the conditional probabilities through a simple logistic predictor, and we use kernel regression. Probability density function is considered in our approach, while they do not consider it. Note that despite all these differences, our approach can be easily modified to be used in classification problems.

Acknowledgement

The authors would like to thank Dr. David Cohn for the helpful discussions and suggestions. We would also thank the reviewers who help us to improve the quality of the paper. This work is supported in part by NSF CAREER Award 9984858.

References

[1] Yong Rui, Thomas S. Huang, and Shih-Fu Chang, "Image Retrieval: Past, Present, and Future", *Proceeding of International Symposium on Multimedia Information Processing*, Dec. 1997.

[2] Donna Harman, "Relevance Feedback Revisited", *Proceedings of the Fifteenth Annual International ACM SIGIR conference on Research and development in information retrieval*, pp. 1-10, 1992.

[3] Gerald Salton and Chris Buckley, "Improving Retrieval Performance by Relevance Feedback", *Journal of the American Society for Information Science*, pp. 288-297, Vol. 41, No. 4, 1990.

[4] Yong Rui, Thomas S. Huang, Michael Ortega, and Sharad Mehrotra, "Relevance Feedback: A Power Tool for Interactive Content-based Image Retrieval", *IEEE Trans. On Circuits and Systems for Video Technology*, pp. 644-655, Vol. 8, No. 5, Sep. 1998.

[5] Y. Ishikawa, R. Subramanya, and C. Faloutsos, "Mindreader: Query Database through Multiple Examples", *Proceeding of the 24th VLDB Conference*, New York, 1998.

[6] Yong Rui and Thomas S. Huang, "Optimizing Learning in Image Retrieval", *Proceeding of IEEE int. Conf. On Computer Vision and Pattern Recognition*, Jun. 2000.

[7] Qi Tian, Pengyu Hong, Thomas S. Huang, "Update Relevant Image Weights for Content-based Image Retrieval Using Support Vector Machines", Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on, pp. 1199-1202, Vol. 2 , 2000.

[8] Sanghoon Sull, Jeongtaek Oh, Sangwook Oh, S. Moon-Ho Song, Sang W. Lee, "Relevance Graph-based Image Retrieval", *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, pp. 713 –716, vol. 2, 2000.

[9] Nikolaos D. Doulamis, Anastasios D. Doulamis and Stefanos D. Kollias, "Non-linear Relevance Feedback: Improving the Performance of Content-based Retrieval Systems", *Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on*, pp. 331-334, Vol. 1, 2000.

[10] T. P. Minka and R. W. Picard, "Interactive Learning Using a 'Society of Models'", *M. I. T Media Laboratory Perceptual Computing Section Technical Report No. 349*.

[11] Ingemar J. Cox, Matt L. Miller, Thomas P. Minka, Thomas V. Papathomas, and Peter N. Yianilos, "The Bayesian Image Retrieval System, PicHunter: Theory, Implementation, and Psychophysical Experiments", *IEEE Trans. On Image Processing*, pp. 20-37, Vol. 9, No. 1, Jan. 2000.

[12] Catherine S. Lee, Wei-Ying Ma and Hong Jiang Zhang, "Information Embedding Based on User's Relevance Feedback for Image Retrieval", Invited paper, SPIE Int. Conf. Multimedia Storage and Archiving Systems IV, Boston, 19-22, Sep. 1999.

[13] Ye Lu, Chunhui Hu, Xingquan Zhu, Hong Jiang Zhang, Qiang Yang, "A Unified Framework for Semantics and Feature Based Relevance Feedback in Image Retrieval Systems", *ACM Multimedia*, 2000.

[14] W. Y. Ma and B. S. Manjunath, "Texture Features and Learning Similarity", *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*, pp. 425-430, 1996.

[15] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby, "Selective Sampling Using the Query by Committee Algorithm", *Advances in Neural Information Processing Systems*, Cambridge, MA: MIT Press, 1993.

[16] Eric Paquet and Marc Rioux, "Nefertiti: A Query by Content Software for Three-dimensional Models Databases Management", *3-D Digital Imaging and Modeling, 1997. Proceedings. International Conference on Recent Advances in*, pp. 345-352, 1997.

[17] Sylvie Jeannin, Leszek Cieplinski, Jens Rainer Ohm, Munchurl Kim, *MPEG-7 Visual part of eXperimentation Model Version 7.0*, ISO/IEC JTC1/SC29/WG11/N3521, Beijing, July 2000.

[18] Titus, Zaharia, François Prêteux, Marius Preda, "3D Shape Spectrum Descriptor", *MPEG-7 ISO/IEC JTC1/SC29/WG11 MPEG99/M5242*, Melbourne, Australia, Oct. 1999.

[19] Motofumi T. Suzuki, Toshikazu Kato, Nobuyuki Otsu, "A Similarity Retrieval of 3D Polygonal Models Using Rotation Invariant Shape Descriptors", *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, pp. 2946-2952, Volume 4, 2000.

[20] Jobst Löffler, "Content-based Retrieval of 3D Models in Distributed Web Databases by Visual Shape Information", *Information Visualization, 2000. Proceedings. IEEE International Conference on*, pp. 82-87, 2000.

[21] Cha Zhang and Tsuhan Chen, "Efficient Feature Extraction for 2D/3D Objects in mesh representation", accepted by ICIP 2001.

[22] H. S. Seung, M. Opper, H. Sompolinsky, "Query by Committee", *Proceedings of the fifth annual ACM workshop on Computational learning theory*, July 27 - 29, Pittsburgh, PA USA, 1992.

[23] Y. Freud, H. S. Seung, E. Shamir, and N. Tishby, "Information, Prediction, and Query by Committee", *Advances in Neural Informations Processing Systems 5*, San Mateo, CA, 1992. Morgan Kaufmann.

[24] Kamal Nigam and Andrew McCallum, "Employing EM in Pool-Based Active Learning for Text Classification", *Proceedings of ICML-98, 15th International Conference on Machine Learning*, 1998.

[25] David D. Lewis and William A. Gale, "A Sequential Algorithm for Training Text Classifiers", *ACM-SIGIR 94*, pp. 3-12, Springer-verlag, London, 1994.

[26] David D. Lewis, "A Sequential Algorithm for Training Tex Classifiers: Corrigendum and Additional Data", *SIGIR Forum*, pp. 13-19, Vol. 29, No. 2, Fall 1995.

[27] Ion Muslea, Steven Minton, and Craig A. Knoblock, "Selective Sampling with Co-testing", *The CRM Workshop on "Combining and Selecting Multiple Models With Machine Learning"*, Montreal, Canada - April 2000.

[28] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan, "Active Learning with Statistical Models", pp. 129-145, Journal of Artificial Intelligence Research 4, 1996.

[29] A. Krogh and J. Vedelsby. *Neural network ensembles, cross validation, and active learning*. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, Cambridge, MA, 1995. MIT Press.

[30] M. Hasenjäger, H. Ritter, and K. Obermayer. *Active learning in self-organizing maps*. In E. Oja and S. Kaski, editors, Kohonen Maps, pages 57-70. Elsevier, Amsterdam, 1999.

[31] Richard O. Duda, Peter E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, 1973.

[32] Alexander G. Gray, Andrew A. Moore, "'N-Body' Problems in Statistical Learning", *Neural Information Processing Systems 2000*, Denver, Colorado, USA, Nov. 2000.

[33] Silverman B.W., *Density Estimation for Statistics and Data Analysis*, New York: Chapman and Hall, 1986.

[34] D. Koller and R. Fratkina, "Using learning for approximation in stochastic processes", *ICML-98*.

[35] A.W. Moore, J. Schneider, and K. Deng, "Efficient locally weighted polynomial regression predictions", *ICML-97*.

[36] S. Thrun, J. Langford, and D. Fox, "Monte Carlo Hidden Markov Models: Learning Non-Parametric Models of Partially Observable Stochastic Processes", *ICML-99*.

[37] Turlach, B.A., "Bandwidth selection in kernel density estimation: A review", *Discussion Paper 9317*, Institut de Statistique, Voie du Roman Pays 34, B-1348 Louvain-la-Neuve, 1993.

[38] S.-T. Chiu, "A Comparative Review of Bandwidth Selection for Kernel Density Estimation", *Statistica Sinica*, pp.129-145, No.6, 1996.

[39] Abramson, I., "On Bandwidth Variation in Kernel Estimates - A Square Root Law", *The Annals of Statistics*, pp. 1217-1223, No. 10, 1982.

[40] Abramson, I., "Arbitrariness of the Pilot Estimator in Adaptive Kernel Methods", *Journal of Multivariate Analysis*, pp. 562-567, No. 12, 1982.

[41] Terrell, G. R. and Scott, D. W. "Variable Kernel Density Estimation", *The Annals of Statistics*, pp. 1236-1265, No. 20, 1992.