ANNOTATING RETRIEVAL DATABASE WITH ACTIVE LEARNING

Cha Zhang and Tsuhan Chen

Dept. of Electrical and Computer Engineering, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
{czhang, tsuhan}@andrew.cmu.edu

ABSTRACT

In this paper, we describe a retrieval system that uses hidden annotation to improve the performance. The contribution of this paper is a novel active learning framework that can improve the annotation efficiency. For each object in the database, we maintain a list of probabilities, each indicating the probability of this object having one of the attributes. This list of probabilities serves as the basis of our active learning algorithm, as well as semantic features to determine the similarity between objects in the database. We show active learning has better performance than random sampling in all our experiments.

1. 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.

Hidden annotation appears to be a powerful tool for bridging the gap between low-level features and high-level semantics. It also powers the system with functionalities such as query with keywords. Early work by Picard and Minka [2] let the user annotate texture pattern in images and extended the knowledge with clustering algorithms. Cox et al. did some experiments on hidden annotation in their Bayesian image retrieval system, PicHunter [3], and showed positive results. 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. As the retrieval database can have a huge size, we want to annotate only part of it. Active learning is used to find the subset of objects such that the annotation of them has the best efficiency.

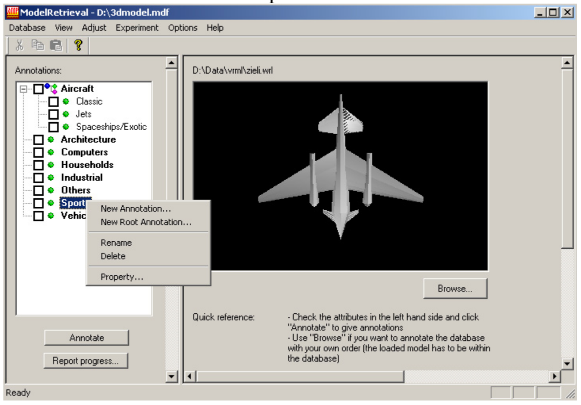
Active learning is a type of machine learning algorithms that can find the statistically "optimal" way to select the training data. While in traditional machine learning research, the learner typically works as a passive recipient of the data, active learning enables the learner to use its own ability 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 [4]-[7]. Recently there has been increasing interest in applying active learning for retrieval systems, such as the work in [8][9]. Both work used support vector machines [10] as the tool for object classification.

This paper is organized as follows. In Section 2, we introduce the general criterion for active learning in our approach. Section 3 presents the details of the proposed algorithm. We discuss the joint-semantic-low-level feature similarity measurement in Section 4. We show the experimental results in Section 5 and conclude the paper in Section 6.

2. THE ACTIVE LEARNING FRAMEWORK

2.1. 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.



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.

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.

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.

2.2. The General Criterion to Choose the Samples

We need to find a general criterion to measure how much information the annotator's annotation can provide to the system. Let $O_i, i = 1, 2, ..., N$ be the objects in the database, and $A_k, k = 1, 2, ..., K$ 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 3.2. In order to derive the expected information gain when we annotate a certain object, we define an uncertainty measurement as follows:

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

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:

- *If object O_i has been annotated, $U_i = 0$;*
- *If* $P_{ik} = 0.5$, for $k = 1, 2, ..., K$, *i.e., we know nothing about the object, $U_i = U_{max}$;*
- *Given* $P_{ik}, k = 1, 2, ..., K$, *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:

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

where E represents the entropy function. We will define uncertainty measurement for multiple attributes in Section 3.3.

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:

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

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 3.1; 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.

3. 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.

3.1. 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. Although many other algorithms are available, in the current system, we use the kernel density estimator [11] simply because the kernel method is also used in updating the probability lists in the next subsection. 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, $r_i = r$, 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:

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

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*.

3.2. Update the Probability Lists

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

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

where $P^{(k)}$ is the prior probability for an object to have attribute A_k. 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. 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. We then extend the knowledge of annotation through *biased kernel regression*. 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 follows:

$$P_{mk} = \begin{cases} 1, & \text{if the object corresponding to } \tilde{\mathbf{x}}_m \text{ has } A_k, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

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

$$P(\mathbf{x} \in A_k) = \frac{\sum_{m=1}^{M} w_m P_{mk} + w_0 P^{(k)}}{\sum_{m=1}^{M} w_m + w_0} \tag{7}$$

where the weights are:

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

Here r_m' is the bandwidth for object $\tilde{\mathbf{x}}_m$. w_0 is a weight representing the confidence of the prior probability.

3.3. 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 2.2, this model is the one that produces the maximum knowledge gain. In order to calculate the gain, we need to define the uncertainty measurement and the probability density for each model.

In section 2.2, 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:

$$U_i = \Psi(P_{i1}) = E(P_{i1}) = -P_{i1} \log P_{i1} - (1 - P_{i1}) \log(1 - P_{i1}) \tag{9}$$

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 real case, we have multiple attributes in the database. We first define individual entropy for an object O_i with respect to attribute A_k as:

$$E_{ik} = -P_{ik} \log P_{ik} - (1 - P_{ik}) \log(1 - P_{ik}). \tag{10}$$

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

$$U_i = \sum_{k=1}^{K} w_{Sk} E_{ik} \tag{11}$$

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 (defined in Section 2.1). The weights are defined as:

$$w_{Sk} = \alpha^{\ell_k - 1} \tag{12}$$

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

With the uncertainty measure in Equation (11) and the probability density estimate in Section 3.1, 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.

4. 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 our system, each model has a list of probabilities of having the attributes, including the query model the user provides. 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:

$$d_{S12} = \sum_{k=1}^{K} w_{Sk} \left[P_{1k}(1 - P_{2k}) + P_{2k}(1 - P_{1k}) \right] \tag{13}$$

where K is the total number of attributes, w_{Sk} is the semantic weight for each attribute as defined in (12), and P_{1k} and P_{2k} are the attribute probabilities for the two models.

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

$$d_{L12} = \sqrt{\sum_{j=1}^{J} w_{Lj} (f_{1j} - f_{2j})^2} \tag{14}$$

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:

$$d_{Overall12} = w_S \cdot d_{S12} + w_L \cdot d_{L12} \tag{15}$$

where w_S and w_L are the semantic weight and the low-level feature weight respectively and $w_S + w_L = 1$.

5. EXPERIMENTS

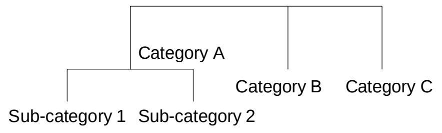


Figure 2 The attribute tree of the synthetic database.

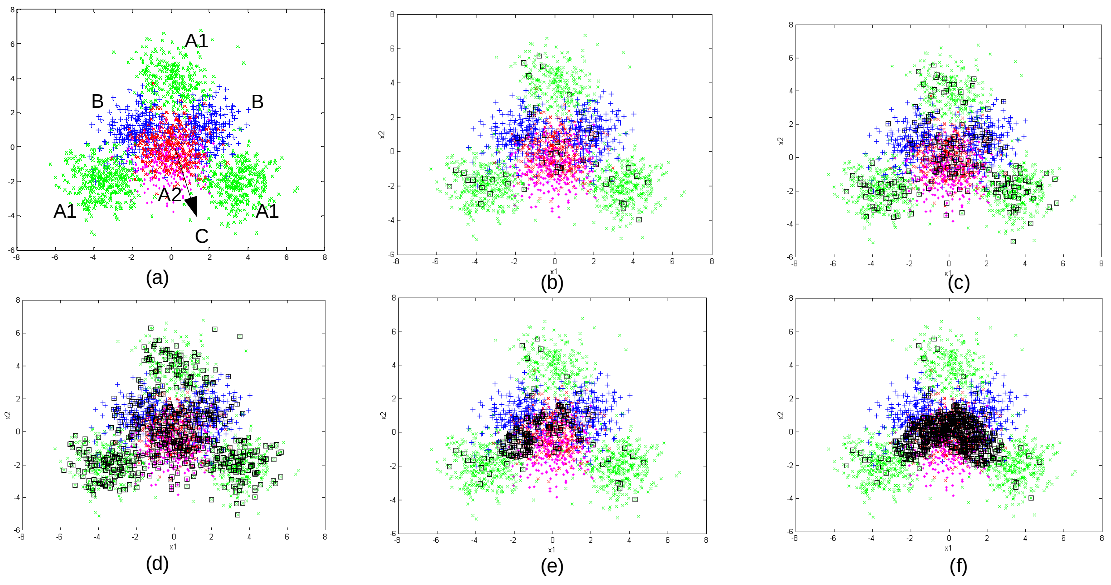


Figure 3 The annotation process of the synthetic database. (a) The categories. (b) Initial 50 sample objects random chosen. (c) Random sampling, 200 objects annotated. (d) Random sampling, 600 objects annotated. (e) Active learning, 200 objects annotated. (f) Active learning, 600 objects annotated.

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 2. The dimension of the low-level feature space is two. The distribution of the categories in the feature space is shown in Figure 3 (a). From Figure 3 (a) we can see that the features of Category B, Category C and Category A Subcategory 2 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 initial the sampling by 50 sample annotations randomly chosen (Figure 3 (b), annotated objects marked with black squares). Figure 3 (c) and (d) shows that under random sampling, the annotated objects are uniformly distributed. Figure 3 (e) and (f) shows that active learning makes the samples focus on the region where the classification is difficult.

We also tested the algorithm on a real database with about 2000 3D models. Positive results are also reported. Due to the page limit of this paper, we refer the interested reader to [12].

6. CONCLUSIONS

In this paper, we proposed a general approach to make hidden annotation with active learning for information retrieval. 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.

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] R. W. Picard and T. P. Minka, "Vision texture for annotation", ACM/Springer Verlag Journal of Multimedia Systems, pp. 3-14, Vol. 3, 1995.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] Simon Tong and Edward Chang, "Support vector machine active learning for image retrieval", *ACM Multimedia* 2001.

[9] Milind Naphade, Ching-Yung Lin, John R. Smith, Belle L. Tseng, and Sankar Basu, "Learning to Annotate Video Databases," *SPIE Electronic Imaging 2002 - Storage and Retrieval for Media Databases*, San Jose, January 2002.

[10] Nello Cristianini, John Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press, 2000.

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

[12] C. Zhang and T. Chen, "An Active Learning Framework for Content Based Information Retrieval", Technical Report, CMU-AMP-01-04.