Hierarchical Shape Classification Using Bayesian Aggregation

Zafer Barutcuoglu* Christopher DeCoro†

Computer Science Department
Princeton University

Abstract

In 3D shape classification scenarios with classes arranged in a hierarchy from most general to most specific, the use of an independent classifier for each class can produce predictions that are inconsistent with the parent-child relationships of the hierarchy. To be consistent, an example shape must not be assigned to a class unless it is also assigned to its parent class. This paper presents a Bayesian framework for combining multiple classifiers based on a class hierarchy. Given a set of independent classifiers for an arbitrary type of shape descriptor, we combine their possibly inconsistent predictions in our Bayesian framework to obtain the most probable consistent set of predictions. Such error correction is expected to improve accuracy on the overall classification by utilizing the structure of the hierarchy. Our experiments show that over the 170-class hierarchical Princeton Shape Benchmark using the Spherical Harmonic Descriptor (SHD)[1], our algorithm improves the classification accuracy of the majority of classes, in comparison to independent classifiers. Our method is also more effective than straightforward heuristics for correcting hierarchical inconsistencies.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques

Keywords: Shape classification, shape descriptors, Bayesian networks, hierarchical classification, machine learning

1 Introduction

A common problem in shape analysis involves assigning semantic meaning to geometry. More generally, given an example shape, it is useful to classify that shape into a pre-existing set of categories, so as to relate it to similar objects. Applications vary widely, ranging from protein binding sites in molecular biology, to object recognition in computer vision. Frequently, the existing classes are specified by example. The classification algorithm is given a "training set" of human-labeled examples for each class, and charged with the task of assigning novel examples to particular classes in a way consistent with the training examples. In addition to specifying the classes themselves, an application may define relationships among classes, commonly in the form of a general-to-specific hierarchy. We present the method of Bayesian Aggregation for the classification of shapes into a hierarchical set of classes, and the main contribution of our work is to take advantage of the relationships represented by the hierarchy to improve classification. We show an example of this in Figure 1. Using a common classification algorithm (Section 2), the eagle model was not classified to its correct class of `flying_bird`, and was incorrectly classified into the class `dragon`. Our method (Section 3) corrects this prediction, giving more accurate results (Section 4).

Given a hierarchical classification H, any class $C \in H$ represents a type of shape, usually with a human-intuitive label (such as "airplane", "animal", etc.), and may have a "parent" class in the hierarchy $P = \text{parent}(C)$ that generalizes the class, as well as an arbitrary number of "descendant" classes $D_i = \text{children}(C)$ that divide

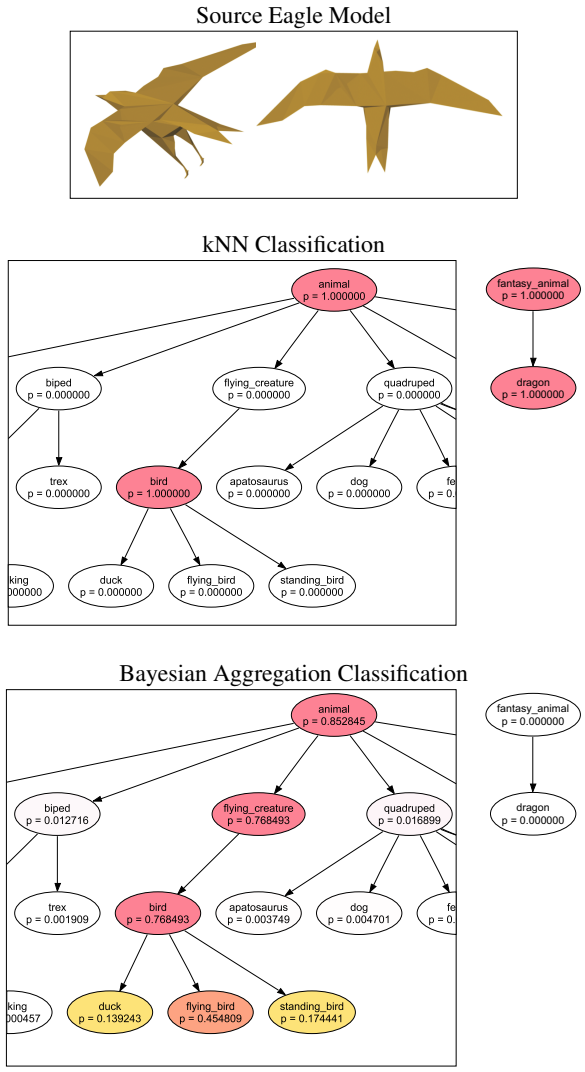


Figure 1: *This model of an eagle from the Princeton Shape Benchmark should be correctly classified as a* `flying_bird`*, and for consistency, it should also be classified as a member of all of* `flying_bird`*'s ancestor classes. Performing the standard k-Nearest-Neighbors algorithm using the Spherical Harmonic Descriptor correctly classifies the model as a bird and an animal, but not as a flying bird or a flying creature; it also incorrectly identifies the eagle as a fantasy animal and a dragon. By applying our Bayesian Aggregation algorithm on the results of the kNN classifiers, we are able to produce a more accurate prediction for the* `flying_bird` *class, as well as resolve the inconsistency with the* `flying_creature` *class, while ruling out the possibility that the model is a dragon. The color of each node represents the probability (p) that the model belongs to that node.*

*e-mail:zbarutcu@cs.princeton.edu
†e-mail:cdecoro@cs.princeton.edu

[1] Excluding 28 isolated nodes from the full 198-class hierarchy

C into more specific classifications. Therefore, for a specific example shape $S \in C$, this implies that $S \in \text{parent}(C)$. Note that for our application, we do not assume that the opposite is true, namely that $S \in C$ does not imply that there exists some $D_i \in \text{children}(C)$ such that $S \in C_i$, or rather, that the children of C do not necessarily partition C.

Given a hierarchy H, and a set of labeled examples in H, a common application is to classify a unique example shape S into the hierarchy. A straightforward method is to train independent classifiers for each possible class in H, and assign S to the classifier that predicts membership of S with the highest confidence. This approach, however, might lead to inconsistencies in the hierarchy, such that the classification of S may violate the rule $S \in C \rightarrow S \in \text{parent}(C)$. While one may choose any number of *ad-hoc* approaches to rectifying this discrepancy, we propose our method, which we term Bayesian Aggregation, to resolve conflicts in a principled manner, and in doing so, improve the overall accuracy of the classification by taking advantage of relationships present between classes in the hierarchy.

2 Background

Shape Descriptors: Our method relies on the ability to first convert an arbitrary geometric model into a vector in R^n, such that for any pair of such vectors, which are referred to as *shape descriptors*, the L^2 metric provides a meaningful measure of similarity. We then use these shape descriptors as feature vectors in our classification algorithm.

For our work, we present results using the Spherical Harmonic descriptor (SHD) [5, 6]. This method has been shown empirically to perform well for the task of shape matching; see for example the applications of SHD in [4, 7, 10]. We stress that our method is independent of the particular shape descriptor used, so long as it maintains the L^2 metric, though classification accuracy will naturally be higher for more discriminating descriptors.

Although these shape descriptors have generally been used for matching tasks, here we use them for classification. Matching only involves retrieving from a set of shapes the ones most similar to a given shape. Classification starts with a set of shapes that are pre-tagged with class labels, and aims to predict the most probable class for a newly presented shape. In this sense, classification extends matching by assigning semantic meaning to shapes and their similarities. For our training and testing examples, we use the models, descriptors and class assignments as provided in the Princeton Shape Benchmark [10]. This has 1814 models in 198 classes, arraigned in a hierarchy that is up to 4 levels deep.

Classification: Many algorithms have been presented in the machine learning literature for classifying unique examples based on a given set of training examples [9]. In general, some method is used to extract a feature vector that concisely represents the example to be classified, and it is the feature vector that is used to represent the example itself. In our algorithm, the shape descriptors are used as feature vectors.

One common method is k-Nearest Neighbors (kNN), which given a feature vector for an example to be classified, finds the k nearest feature vectors from the labeled training set, and assigns to the example the most common among the k labels [9]. The value k acts as a smoothing parameter, and larger values of k will tolerate more noise in the training set. In the case of $k = 1$ (1-NN) this is equivalent to assigning the label of the nearest training example. The assumption of kNN is that similarity between examples is represented accurately by Euclidean distance of their feature vectors, for which our shape descriptors are designed, making kNN a reasonable choice as a basic independent classifier. We then show how Bayesian Aggregation improves the results of kNN.

There exist other, more sophisticated, methods of classification where different assumptions are made on the distribution of the data. Popular methods include support vector machines [3], which find the optimal linear separating plane for a kernel-transformed feature space, corresponding to different distributions of data. Another popular method is artificial neural networks [2], which are a general form of a non-linear function that can be manipulated to create arbitrary decision boundaries. Any such method can be used as the base classifier for our technique, though we use kNN in our experiments. One of the authors has previously used techniques similar to those presented in this work for the task of protein function prediction [1]. We have found that our results for shape classification are in fact better than those for function prediction.

3 Algorithm Description

We perform hierarchical classification of shapes by first using a set of independently trained classifiers (called the "base" classifiers) to predict binary memberships for each class in the hierarchy. This is known as the "base" classification. Then we construct a Bayesian hierarchical combination scheme which performs collaborative error correction over their possibly-inconsistent predictions. A Bayesian network involves a number of random variables, some of which are observed directly, while others are hidden. Of the hidden variables, some are assumed to be conditionally dependent on other variables. We can visually represent this as a graph, as in Figure 2. Nodes represent variables, and edges represent conditional dependence. Given values for observed nodes, Bayesian inference algorithms use this network to assign probabilities for hidden nodes, given values for observed nodes or find the most probable set of consistent labels given the initial predictions.

For a given example, let y_i denote the actual binary membership to class i, and $\hat{y}_i$ denote the base classifier prediction for that class[2]. After obtaining a set of (possibly inconsistent) binary $\hat{y}$ predictions from all base classifiers, we wish to find the most probable set of (consistent) y labels that may be underlying them. Therefore, for N nodes we need to find the labels $y_1 \ldots y_N$ that maximize the conditional probability $P(y_1 \ldots y_N | \hat{y}_1 \ldots \hat{y}_N)$, which by Bayes rule equals

$$\frac{P(\hat{y}_1 \ldots \hat{y}_N | y_1 \ldots y_N) P(y_1 \ldots y_N)}{Z}, \tag{1}$$

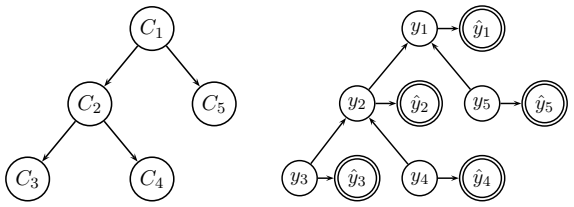
where Z is a constant normalization factor. We propose a Bayesian network structure for this problem, as illustrated by Figure 2. The class hierarchy shown at left is transformed into a Bayesian network by adding extra nodes that correspond to the observed classifier outputs. The y-nodes are probabilistically dependent on their child classes, and the $\hat{y}$-nodes are probabilistically dependent on their corresponding labels y.

We enforce hierarchical consistency of labels using the edges among the y-nodes. Each edge represents a conditional probability table, and we set them to ensure that a label is 1 when any one of its children is 1. The edges from y to $\hat{y}$ reflect an important observation: for a given example, a classifier prediction $\hat{y}_i$ is independent of all other classifiers' predictions $\hat{y}_j$ and labels y_j ($i \neq j$) given its true label y_i. This simplifies Equation 1, since we can write

$$P(\hat{y}_1 \ldots \hat{y}_N | y_1 \ldots y_N) = \prod_{i=1}^{N} P(\hat{y}_i | y_i). \tag{2}$$

$P(\hat{y}_i | y_i)$ consists of $P(\hat{y}_i | y_i = 1)$ and $P(\hat{y}_i | y_i = 0)$, which are binomial distributions for our binary predictions, and can be estimated during training by validation. We use cross-validation to hold out part of the training data and evaluate them using the rest,

[2] Although for this application we use base classifier predictions that are binary, our method easily generalizes to base classifiers with real-valued predictions.



(a) Hierarchy of classes (b) Bayesian network

Figure 2: *The class hierarchy* (a) *is transformed into a Bayesian network* (b)*. The y nodes are the binary-valued hidden nodes representing actual membership to the class, and the corresponding $\hat{y}$ nodes are the observed classifier outputs.*

which produces a 2×2 *confusion matrix* of true-positive (TP), true-negative (TN), false-positive (FP) and false-negative (FN) counts, from which we can directly infer the probabilities; e.g. $P(\hat{y}_i = 1 | y_i = 1) = TP/(TP+FN)$.

Since each label y-node is conditionally dependent only on its children, we can also make the following simplification

$$P(y_1 \dots y_N) = \prod_{i=1}^{N} P(y_i | \vec{y}_{children(i)}), \tag{3}$$

where $\vec{y}_{children(i)}$ is used to denote all child y-nodes of node y_i, and $P(y_i | \vec{y}_{children(i)})$ can be straightforwardly inferred from the training set by counting.

Now that the Bayesian network is defined, any standard Bayesian inference algorithm can be used to find the most likely configuration of (consistent) hidden y labels for the given $\hat{y}$ predictions, or the marginal distribution $P(y_i | \hat{y}_1 \dots \hat{y}_N)$ for each class separately. We use the marginal probabilities $P(y_i = 1 | \hat{y}_1 \dots \hat{y}_N)$ in our results so that we retain real-valued membership probabilities and can threshold them at different levels as desired. Among the many Bayesian inference algorithms available, in our experiments we used the *junction tree* algorithm for exact inference, although approximate inference with Monte Carlo methods such as *Gibbs sampling* may be more feasible for more complex hierarchies. Descriptions and detailed references for these and other inference algorithms are available in [8].

4 RESULTS

We evaluated our method by first performing two-fold cross-validation[3] on each class using *k*NN. The value of k for each class was chosen as the best $k \in \{1,3,5,7,9\}$ using leave-one-out cross-validation accuracy[4]. Then we combined the held-out confusion matrices from both folds to build our Bayesian network for the hierarchy, and computed the marginal membership probability of each example for each class using Bayesian inference.

The predictions from the initial *k*NN classifiers are binary (for each class, we consider each example in the training set strictly as either a positive or negative example) whereas the marginal probabilities from the Bayesian network are real-valued. This allows us to threshold the Bayesian probabilities at different thresholds to obtain different binary predictions according to the risk model of different applications. For example, a particular application might have a very high cost for false-positives while false-negatives might be less of a concern, so in that case we would choose a high probability threshold and get positive predictions that are rare but confident. Instead of evaluating with a particular threshold, we consider an evaluation over all possible thresholds. The *ROC curve* for a real-valued classifier is a plot of the true-positive ratio $TP/(TP+FN)$ against the false-positive ratio $FP/(TN+FP)$ for all possible thresholds, so it connects $(0,0)$ to $(1,1)$. The area under the ROC curve, called the *AUC score*, ranges from 0.5 (random guessing) to 1 (a perfect classifier), and allows us to evaluate the classifier's performance over all possible thresholds.

Since the original *k*NN classifiers have binary outputs, they correspond to a single point on the ROC axes instead of a curve. However, for any two points (classifiers) on an ROC graph, one can obtain any point on the connecting line segment by randomly selecting between the two classifiers' predictions with a Bernoulli distribution, so we compare the convex hull of the Bayes-net ROC curve to the "convex hull" of the naked *k*NN, which is its single point connected to $(0,0)$ and $(1,1)$.

The mean AUC score over the 170 independent *k*NN classifiers was 0.7004. 27 of these had an AUC of 0.5, meaning they were no better than random guessing. The mean AUC score of marginal Bayesian probabilities was 0.8369, reflecting a mean AUC improvement of 0.1365. All classes improved in AUC score, with the exception of one class (`torso`) which stayed the same at 0.9994. The scatter-plot of AUC scores before and after Bayesian aggregation is shown in Figure 3.

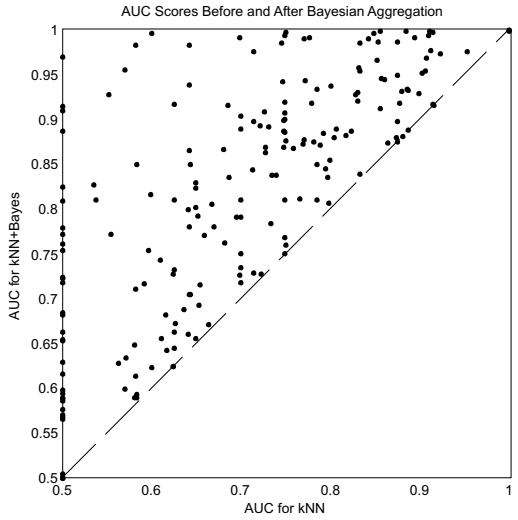


Figure 3: *Per-class improvement in AUC score. For each class, we compare the AUC score of simple kNN versus the AUC score for the class after the application of Bayesian Aggregation. The dashed line represents no change in AUC, and the values range from* 0.5 *(random guessing) to* 1.0 *perfect accuracy. As the figure shows, the use of Bayesian Aggregation results in a significant improvement in classification accuracy.*

Figure 1 shows an illustrative set of predictions for the example **m42**, which is a model of an eagle. Independent *k*NN classifier predictions include an inconsistency at node `flying_creature` and false positives for `fantasy_animal` and `dragon`. Bayesian hierarchical aggregation yields $p = 0$ for the false positives, correctly fixes the inconsistency at `flying_creature`, and further yields $p = 0.454809$ for the leaf `flying_bird` of which **m42** actually is a member. Its sibling leaf nodes `standing_bird` ($p = 0.174441$) and `duck` ($p = 0.139243$) have significantly lower probability. After those, the next highest probability in the entire hierarchy is very low ($p = 0.038353$). This example demonstrates that our method extends beyond only fixing inconsistencies and is able to improve

[3] The scarcity of positive examples in many classes prevented us from using a larger number of folds.

[4] 134 of 170 classes chose $k = 1$, and none chose $k = 9$.

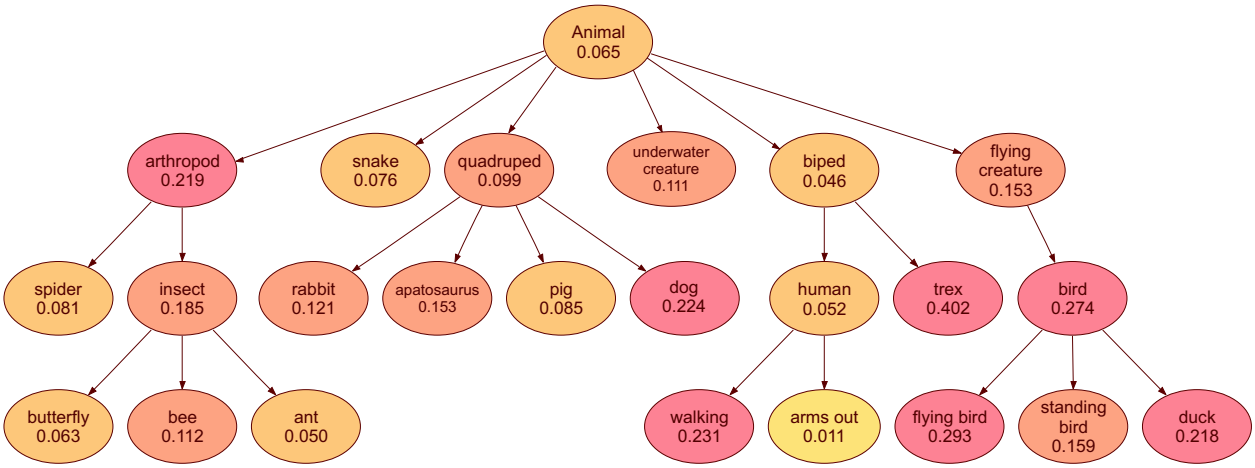


Figure 4: *Hierarchical improvement in AUC score. For the animal branch of the hierarchy, we show the improvement in AUC score for each node, after applying our method. The darker nodes are those with larger increases in AUC score. As the figure shows, all the nodes improved, many by significant amounts. This is only one slice of the entire Princeton Shape Benchmark, but other branches display similar results.*

classifiers anywhere in the hierarchy. For further examples of this, we show a portion of the hierarchy in Figure 4, and the amount of improvement in AUC score for each node. All of the nodes shown improved, many by significant amounts. Note for example the improvement in the `bird` branch, which corresponds to the classifiers used for Figure 1.

To evaluate the increase in classification accuracy that is due to our method, we compare our results to two straightforward methods for conflict resolution. One such method propagates negative predictions downward, so that if a node predicts that an example is not a member, its children are also forced to predict that the example is not a member. While this resolves inconsistencies, it shows very poor performance: 88 of the classes decreased in performance, and the average change in AUC score is -0.05. Another method propagates positive predictions upward, increasing the average AUC score by only 0.01, which is significantly less than the improvement due to our method.

Note that our method is more general than the problem at hand, and does not enforce the single-branch nature of the Princeton Shape Benchmark dataset; i.e., it allows multiple-branch predictions. A post-processing step to convert them into single-branch predictions would be appropriate in a practical application, and could further improve our results, but we chose to leave out that constraint in this evaluation to keep comparisons to *k*NN fair.

5 Conclusions and Future Work

Independent classifiers for a hierarchy of classes can violate the hierarchy in their predictions. Our Bayesian hierarchical aggregation algorithm corrects such inconsistencies, and substantially improves performance of classifiers over the whole hierarchy by allowing connected classes to influence one another. This method is directly applicable to any type of binary or real-valued base classifier and any hierarchy in the form of a directed acyclic graph.

For applications with more training data, simpler, more concise representations such as support vector machines might prove to be more applicable than *k*NN classifiers. However, straightforward linear SVMs might not have the discriminating ability of *k*NN, due to the ability of *k*NN to form non-linear decision boundaries. It is possible that SVMs with more complicated, non-linear kernels would also be applicable, though automated parameter selection for every class in the hierarchy would be computationally expensive. Other future areas of investigation include exploring the effect of various shape descriptors, and using machine learning techniques to evaluate the discriminative quality of each descriptor.

We showed that in contrast to Bayesian aggregation, straightforward methods for resolving inconsistencies do not significantly improve classification, and may in fact decrease performance. Top-down conflict resolution places excessive responsibility on top-level nodes, reducing the ability of low-level nodes to affect classification. Bottom-up conflict resolution allows the more general parent nodes to rely on the more specific predictions of the child nodes, slightly improving the performance in those cases where inconsistencies exist, and in these classes only. Our method improves classification accuracy not only for these cases, but even in the absence of inconsistencies, and exceeds the improvements of straightforward methods by an order of magnitude.

Acknowledgments

We would like to thank Philip Shilane and Szymon Rusinkiewicz of Princeton University for their invaluable comments and feedback on this work.

References

[1] Zafer Barutcuoglu, Robert E Schapire, and Olga G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 2006.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

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

[4] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, and D. Jacobs. A search engine for 3D models. *ACM Transactions on Graphics (TOG)*, pages 83–105, January 2003.

[5] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Symposium on Geometry Processing*, pages 167–175, 2003.

[6] Michael Kazhdan. *Shape Representations and Algorithms for 3D Model Retrieval*. PhD thesis, Princeton University, 2004.

[7] Patrick Min. *A 3D Model Search Engine*. PhD thesis, Princeton University, 2003.

[8] Kevin P. Murphy. The Bayes Net Toolbox for MATLAB. *Computing Science and Statistics*, 33, 2001.

[9] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 2nd ed.* Prentice Hall, December 2002.

[10] P. Shilane, M. Kazhdan, P. Min, and T. Funkhouser. The Princeton shape benchmark. In *Proc. Shape Modeling International, Genoa, Italy*, 2004.