SHREC'08 Entry: Training Set Expansion via Autotags

Corey Goldfeder Haoyun Feng Peter Allen

Columbia University

ABSTRACT

Training a 3D model classifier on a small dataset is very challenging. However, large datasets of partially classified models are now commonly available online. We use an external training set of models with associated text tags to automatically assign tags to both training and query models. The similarity between these tags, used in conjunction with a standard shape descriptor, yields a multiclassifier that outperforms the standalone shape descriptor.

KEYWORDS: Classification, labeling, autotagging.

INDEX TERMS: H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing

1 INTRODUCTION

In this SHREC entry we use an existing shape descriptor and standard k-Nearest Neighbor classification. Our contribution lies in automatically providing text labels for both the training and query models and using the resulting tags alongside geometry as a multiclassifier. Text descriptions can link models that belong to the same semantic class but have significantly different geometry; if an airplane and a helicopter are both tagged as "aircraft" we can link them as belonging to the same class even if they are geometrically too different to match.

The 3D classification problem is often posed with a very unbalanced selection of training models. The dataset provided for SHREC was a good example of this; at the finest classification the class sizes ranged from as small as 1 model to as large as 21 models, with an average class size of 3.9 and a standard deviation of 3.96. Training a classifier on this sort of data is very challenging, even with a good shape descriptor. Our algorithm produces autotags by appealing to an external training set, which can help us handle very small training classes, as there may be a larger class in the external training data that links a small training class and a query model.

2 TRAINING SET EXPANSION

To classify the query models, we first classify both the query models and all of the training models on a larger training set. We could then use the classes from the larger training set to connect query models to our actual training set – models that were assigned into the same classes using the larger set should be assumed to be in the same classes for the smaller set as well. However, this assumes that the classifications of the larger training set were done in a similar way to the classifications of the smaller set, which is not necessarily the case.

To avoid this problem, we use text tags rather than explicit classes on the larger set and automatically propagate a set of possible tags to the training and query sets. There is no requirement that the tags on the training class neatly break down into classes; a model can have multiple overlapping tags that do not fit into a hierarchy. There is also no requirement that the tags be entirely trustworthy. We can assign an initial probability to a tag and model pair representing our confidence in the appropriateness of that tag for that model.

2.1 Autotagging

Our approach is to augment the training set using an external set of tagged models. To do this, we make use of our *autotagging* algorithm [1] which uses a corpus of known tagged models $\Omega = \{\omega_1, \omega_2 ... \omega_x\}$ to probabilistically propagate text labels drawn from the set of all possible labels $\Lambda = \{\lambda^1, \lambda^2 ... \lambda^x\}$ to an untagged model ω. We start with a geometric shape distance (we use Zernike descriptors [2]) and find the k nearest neighbors of ω within some threshold. The distances are normalized to lie in the range $(0, 1)$. We take

$$P(\omega_x \approx \omega_y) = (1 - D(\omega_x, \omega_y))^2 \quad (1)$$

to be an estimate of the probability that ω_x and ω_y represent the same type of object and therefore should have similar text tags. Then given our untagged model ω, a possible text tag λ^i, and a neighbor ω_x from the corpus, the probability that our query model should have the tag, is

$$P(\lambda^i, \omega) = P(\omega \approx \omega_x) \wedge P(\lambda^i, \omega_x). \quad (2)$$

Intuitively this means that the probability that λ^i is appropriate for ω is the probability that it is appropriate for ω_x and that ω and ω_x are similar enough to share tags. (We can think of $P(\lambda^i, \omega_x)$ as a measure of how much we trust the original annotation of the corpus.) When considered over the full set of k neighbors this generalizes to

$$P(\lambda^i, \omega) = \bigcup_{n=1}^{k} P(\omega \approx \omega_n) \wedge P(\lambda^i, \omega_n). \quad (3)$$

By analogy to the tf-idf method from text processing [3] we reweight these probabilities such that

$$P_{tf-idf}(\lambda^i, \omega_x) = \frac{P(\lambda^i, \omega_x)}{\sum_j P(\lambda^j, \omega_x)} \cdot \log \frac{|\Omega|}{\sum_{\omega_k \in \Omega} P(\lambda^i, \omega_k)}. \quad (4)$$

For each untagged model, we get a vector of probabilities for each tag. Once we have these tag vectors, they can be compared using either the cosine between them or their Euclidean distance; we chose to use the latter.

2.2 Autotagging Implementation

In our SHREC entry we used a corpus containing 1959 3D models, 1814 of which were taken from the Princeton Shape Benchmark (PSB) [4] and 145 of which were the pre-training set distributed by the organizers several days before the competition. For each PSB model, we parsed the included classification of the PSB and tagged the model with the names of each class it belonged to. For example, a commercial airliner was tagged as {"aircraft", "airplane", "commercial"}, corresponding to the three classes that it belongs to in the PSB. The remaining 145 models were hand tagged with text labels.

For each model in the corpus we computed a Zernike descriptor with 20 levels of moments, on a voxel grid with 128 voxels to a side and with a binary thickening kernel of 4. The resulting 121 dimensional vectors were then transformed via PCA onto a new basis such that more variance was packed into the first few dimensions. As we did not discard any dimensions, the PCA transform does not affect the results. We used a hypercube-pruned projection search [5] over a PostgreSQL database to find nearest neighbors. Packing most of the variance into a few dimensions allows us to search higher-variance dimensions first and prune most datapoints very early in the search.

3 CLASSIFICATION

In a training stage, we autotag each of the 425 models in the training set. To classify a query model we first compute its autotags. We then calculate two distances between the query model and each training model; the Zernike descriptor distance, and the Euclidean distance between the weighted autotag vectors. We normalize both sets of values to zero mean, unit variance and combine them into a single multiclassifier distance using a weighted average. (Our experiments confirmed the results of Min et al. [6] that a linear weighted average with geometry weighted by 0.9 and text distances weighted by .1 produced a good multiclassifier.)

We use a *k*-Nearest Neighbor approach to classification. We find the distances to the *k* nearest neighbors, and clamp them to lie in the range (-3, 3) which represents 3 standard deviations from the mean. (These "distances" can be negative since we have previously subtracted the mean). We then subtract the distances from the maximum possible distance, which is 3, and treat the results as affinities, with a higher value indicating a more likely classification. The sum of these affinities is normalized to 1, and the affinities for each class are combined, resulting in a final output of positive probabilities for up to *k* classes, and zero probabilities for the remaining classes.

For the medium and fine classifications, it was necessary to use a fairly small value of $k = 3$ in order to avoid drowning out results from the smaller classes. For the coarse classification, which has more evenly balanced classes, we submitted runs with both $k = 3$ and $k = 5$.

4 RESULTS

For each query model our algorithm produces between 1 and k classification attempts, ranked by the confidence our algorithm has that the classification is correct. For the coarse classification (12 classes), and with $k = 3$, the best-ranked attempt was correct 67.1% of the time, the second ranked attempt 78.9% of the time, and the third ranked 82.9%. With $k = 5$ the accuracies were 61.8% for the first attempt, 80.3% for the second, and 85.5% for the third, with no increase in accuracy beyond the third attempt. For the medium classification (39 classes) the accuracies were 60.5%, 65.8%, and 67.1%, while for the fine classification (109 classes) the accuracies were 50%, 55.3%, and 56.6%.

In this work we proposed a distance metric for 3D models consisting of the Euclidean distances between vectors of tags learned from an external corpus. Our claim was that this distance, when combined with a geometric shape distance, yields improved classifications. Figure 1 shows the accuracy of classification with out combined classifier, compared to the accuracy of a classifier that uses only the geometric Zernike distances and ignores tags.

For the coarse classification, and with $k = 3$, our multiclassifier was 2.6% more successful on the first and second ranked attempts and 3.9% more successful on the third attempt. With $k = 5$ our multiclassifier was 3.6% less successful on the first attempt, but

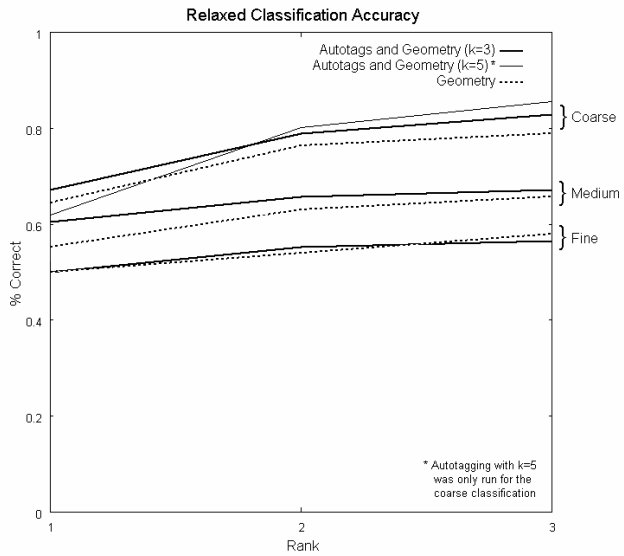

Figure 1. The classification accuracy of our text and geometry multiclassifier, as compared to the geometry classifier alone.

jumped to 6.6% better than the geometric classifier by the third. For the medium classification, our multiclassifier was 5.3% better than the geometric classifier on the first attempt, 2.6% better on the second, and 1.3% better on the third. For the fine classification the performances of the two classifiers on their first attempts were identical, with our multiclassifier performing 1.3% better on the second attempt and 1.3% worse on the third. Overall, our approach provided improved accuracy for the coarse and medium classifications and comparable accuracy for the fine classification.

5 CONCLUSIONS

We have shown that autotag distances can be combined with a geometric classifier to form an improved multiclassifier. Although we used Zernike descriptor distances for the geometric portion of the classifier, our algorithm can in principle be used with any other geometric shape distance, and in future work we would like to examine which shape distances are most appropriate.

In our experiments, we observed that the classification results of our algorithm and the Zernike classifier differed not only in accuracy but also in which models were correctly classified. We correctly classified some models that the Zernike classifier could not, but we also incorrectly classified some models that the Zernike classifier was correct on. In the future we will examine other methods of combining classifiers in the hopes of preserving more of the good classifications from the geometric approach while still adding new good classifications from the tags.

REFERENCES

[1] C. Goldfeder, P. Allen. Autotagging To Improve Text Search for 3D Models. *Joint Conference on Digital Libraries*, 2008. (*to appear*)

[2] M. Novotni, R. Klein. 3D Zernike descriptors for content based shape retrieval. *Solid Modeling and Applications*, 2003.

[3] G. Salton. Term-weighting approaches in automatic text retrieval. *Information Processing and Management* 24(5), 1988.

[4] P. Shilane, P. Min, M. Kazhdan, T. Funkhouser. The Princeton Shape Benchmark. *Shape Modeling International*, 2004.

[5] S. A. Nene, S. K. Nayar. A simple algorithm for nearest neighbour search in high dimensions. *Transactions on Pattern Analysis and Machine Learning* 19(9), 1997.

[6] P. Min, M. Kazhdan, T. Funkhouser. A comparison of text and shape matching for retrieval of online 3D models. *European Conference on Digital Libraries*, 2004.