Discriminative Learning of Relaxed Hierarchy for Large-scale Visual Recognition

Tianshi Gao
Dept. of Electrical Engineering
Stanford University
tianshig@stanford.edu

Daphne Koller
Dept. of Computer Science
Stanford University
koller@cs.stanford.edu

Abstract

In the real visual world, the number of categories a classifier needs to discriminate is on the order of hundreds or thousands. For example, the SUN dataset [24] contains 899 scene categories and ImageNet [6] has 15,589 synsets. Designing a multiclass classifier that is both accurate and fast at test time is an extremely important problem in both machine learning and computer vision communities. To achieve a good trade-off between accuracy and speed, we adopt the relaxed hierarchy structure from [15], where a set of binary classifiers are organized in a tree or DAG (directed acyclic graph) structure. At each node, classes are colored into positive and negative groups which are separated by a binary classifier while a subset of confusing classes is ignored. We color the classes and learn the induced binary classifier simultaneously using a unified and principled max-margin optimization. We provide an analysis on generalization error to justify our design. Our method has been tested on both Caltech-256 (object recognition) [9] and the SUN dataset (scene classification) [24], and shows significant improvement over existing methods.

1. Introduction

Multiclass classification problems are fundamental in computer vision. Many vision tasks, *e.g*. object recognition, multiclass image segmentation and scene classification, require the classifier to discriminate multiple categories. Due to the richness of visual concepts in the real world, the number of categories in these tasks is usually very large. For example, the newly collected SUN dataset [24] has 899 scene categories, ImageNet [6] contains $15,589$ non-empty synsets, and the latest large scale visual recognition challenge requires the algorithm to predict the presence of objects from $1,000$ categories [1].

In this paper, our goal is to design a multiclass classification method that is both accurate and fast when there are a large number of classes. To achieve low computational complexity, we exploit the hierarchical structure in the label space by organizing a set of binary classifiers in a DAG or tree structure. To achieve high classification accuracy while being fast, we adopt the relaxed hierarchy structure from [15], where a subset of confusing classes is allowed to be ignored in each node. The key contribution and novelty of our work is that we learn the coloring of the classes and the induced binary classifier using a unified and principled max-margin optimization. The coupling of the coloring and the learning of the decision boundary directly maximizes the margins of the induced binary problem leading to better performance. Furthermore, we provide a theoretical analysis of the generalization error to justify our design.

2. Background and Related Work

In this section, we review existing methods and discuss their performances in terms of both classification accuracy and test speed, when there are a large number of classes.

The most popular family of multiclass classification algorithms reduces multiclass to binary. We focus on two classes in this family depending on whether the methods take advantage of the hierarchical structure in the label space. The first class is generic and treats the label space as fat. It includes *one-vs-all*, *one-vs-one with Max Wins* (all pairs of classes are compared and the class with the most votes wins), *error correcting output codes* (first proposed in [7] and unified by Allwein *et al.* [1]), *output-coding based multiclass boosting* [19, 12] (learning the coding matrix and the induced binary classifiers in a stage-wise way), and *DAGSVM* [17] (one-vs-one classifiers are organized in a directed acyclic graph to discard classes sequentially). Although these methods have shown good classification accuracy, the computational efficiency is not maximized due to ignoring the hierarchical structure in the label space.

The second class of methods reduces the test time by exploiting the hierarchical structure. The main difference between various existing methods is the way the hierarchy is

[1] http://www.image-net.org/challenges/LSVRC/2010/index

built, which can be divided into two main groups. The first group constructs the hierarchy by recursive top-down partitioning of the set of classes. Chen *et al.* [4] used max-cut based on KL divergence between feature distributions from different classes. Vural *et al.* [22], Liu *et al.* [13] and Yuan *et al.* [25] used k-means in the original feature space. Griffin *et al.* [10] and Bengio *et al.* [3] treated the confusion matrix as an affinity matrix between classes and used spectral clustering to partition the classes. Pujol *et al.* [18] used a greedy sequential forward floating search algorithm to find two subsets of classes with the maximum mutual information between the feature distributions. The second way to construct the hierarchy is by bottom-up agglomerative clustering. Zhigang *et al.* [14] used one class SVM to estimate pairwise distances between classes in the feature space implicitly defined by the kernel and merge classes recursively. Xia *et al.* [23] also used bottom-up merging but using class distances estimated from the original feature distribution. One common fundamental problem for these methods is that the assumption about the good separability of a binary partition of classes at each node is not valid when there are a large number of classes [15]. Thus, these methods do not scale well in terms of the classification accuracy [15]. Escalera *et al.* [8] relaxed the assumption by allowing subclasses within a single class to be considered separately. However, this implies that the equivalent number of classes is even larger than the original number, resulting in high computational complexity. Marszalek *et al.* [15] proposed a better relaxation by delaying the decisions for a subset of confusing classes. The common fundamental problem with all the above methods is the suboptimal two-stage procedure. They first use some measures about the separability of classes to partition, and then find a decision boundary to separate them. However, the measure of separability in the first step is not equivalent to the separation margins achieved by the decision boundary in the second step. For example, some existing methods use k-means to cluster classes in the original feature space but use a non-linear kernel SVM in the next step, some methods partition the classes based only on confusion matrix and so on.

Another type of multiclass classification methods learns the multiple models for all classes jointly, *e.g.*, multiclass SVM [5]. Although multiclass SVM nicely couples the parameters of different classes under a single objective, computationally it is more expensive to train and only scales linearly at test time.

There are several other works learning/using hierarchies. Zweig *et al.* [26] built object hierarchy by hand for a small number of categories and combined models from multiple levels, but it does not scale well. Bart *et al.* [2] and Sivic *et al.* [20] focused on unsupervised discovery of visual hierarchies from a collection of unlabeled images. Li *et al.* [11] also proposed to discover the "semantivisual" image hierarchy by using both image and tag information. These works used generative models to discover the ontology but with less emphasis on the supervised discriminative classification task and its computational property.

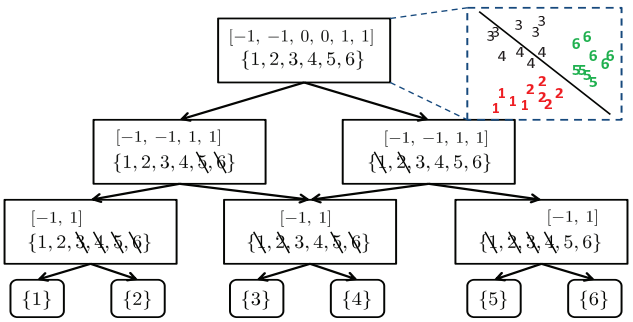

Figure 1. A relaxed hierarchy for a 6-class problem. In each node, the first row is the coloring of the classes, and the second row is the set of remaining classes to consider. The dash box next to the root illustrates the coloring and the decision boundary. Each number represents an instance with its true label, and the red, green and black color the instances into -1, 1 and 0 respectively. The black line represents the decision boundary between the positive and negative classes while ignoring the 0 classes.

3. Model

3.1. Overview

Our goal is to design a model that is both accurate and fast when there are a large number of classes. To achieve fast evaluation, we exploit the hierarchical structure in the label space by organizing a set of binary classifiers in a hierarchy, where each binary classifier considers two subsets of classes. In this way, the label space can be pruned effectively and actively resulting in an efficient instance-specific decision path. To achieve high accuracy while being fast, we adopt the relaxed hierarchy structure from [15], where "relaxed" means that each node can ignore a subset of confusing classes. When there are a large number of classes, if confusing classes are not allowed to be ignored, nodes at a higher level of the hierarchy will have high errors which can never be recovered in subsequent evaluations. As an example, Figure 1 shows a sample relaxed hierarchy for a 6-class problem. We color the classes as one of $\{-1, 0, +1\}$, where classes colored as -1 will be learned to separate from those with color $+1$. Classes with color 0 will be ignored in the node binary classification, so they need to be considered in both the left and right children. Note that at the root node, classes 3 and 4 are ignored, since a full binary partition will lead to a hard binary problem without a perfect separation hyperplane. At test time, an instance traverses the hierarchy from top to bottom until a leaf node where the final decision is made. Note that after each classifier evaluation, a subset of classes will be pruned away, so the number of classifier evaluations is bounded by the number of classes. At training time, we want to learn the hierarchy by learning the coloring and the binary classifier for each node. We

discuss about the learning in the next subsection.

3.2. Learning

Given some algorithm that takes a set of training instances as input, colors the classes and trains the induced binary classifier, one can recursively apply this algorithm to generate a whole hierarchy in a top-down manner. Therefore, the key is to design such an algorithm for learning a single node.

Formulation. So what is a good learning algorithm for a single node? Consider two extreme cases: (1) all instances from all classes participate in the binary problem; (2) only two classes are considered. In terms of the computational complexity, the first case is more efficient, since it's capable of pruning away a large subset of candidate classes after evaluating a single node classifier while the second case can only eliminate one class. This implies that the first case requires fewer classifier evaluations. In addition, for the overall efficiency, i.e., to minimize the height of the hierarchy, a balanced partition is preferred. In terms of the classification accuracy, however, the second case might be better, since it considers instances only from two classes instead of having to induce a decision boundary for all instances from all classes. Therefore, to have a good trade-off between the accuracy and complexity, we want to find a coloring so as to minimize the error of the induced binary classification while maximizing the number of classes participating in the binary problem and preferring balanced partitions.

We now formulate this intuition into a principled optimization problem. At each node, given a set of training instances $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $y_i \in \mathcal{Y} = \{1, 2, \ldots, K\}$, we want to partition the label space $\mathcal{Y}$ into: a positive subset S_y^+, a negative subset S_y^- and an ignore subset S_y^0. A particular partition gives rise to a binary classification problem where the positive and negative instances are from $S_x^+ = \{\mathbf{x}_i | y_i \in S_y^+\}$ and $S_x^- = \{\mathbf{x}_i | y_i \in S_y^-\}$ respectively. We introduce a set of coloring variables $\{\mu_k\}_{k=1}^K$. Each μ_k can take a value from $\{-1, 0, +1\}$ indicating which group class k belongs to. Then we have:

$$
\begin{aligned}
\min_{\mathbf{w}, b, \{\mu_k\}, \{\xi_i\}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^N |\mu_{y_i}|\xi_i - A\sum_{i=1}^N |\mu_{y_i}| \\
\text{subject to} \quad & \mu_k \in \{-1, 0, +1\},\ \forall k \in \mathcal{Y} \\
& \mu_{y_i}(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i,\ \forall i \\
& \xi_i \geq 0,\ \forall i \\
& -B \leq \sum_{k=1}^K \mu_k \leq B \\
& \sum_{k=1}^K \mathbf{1}\{\mu_k > 0\} \geq 1 \text{ and } \sum_{k=1}^K \mathbf{1}\{\mu_k < 0\} \geq 1
\end{aligned} \tag{1}
$$

where $\mathbf{1}\{\cdot\}$ is an indicator function. The variables we are optimizing over can be divided into two parts: (i) $\{\mu_k\}$ which colors the label space; (ii) $\{\mathbf{w}, b, \{\xi_i\}\}$ which defines a decision hyperplane and the hinge loss for the binary classification problem induced by the coloring. For the objective function, there are two components designed to minimize the hinge loss while encouraging more instances (classes) to participate. Specifically, the first component of the objective $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^N |\mu_{y_i}|\xi_i$ is trying to minimize the regularized hinge loss on $S_x^+ \cup S_x^-$. This can be done trivially by setting all μ_k's to be 0. Therefore, we introduce the second component, *i.e.*, $-A\sum_{i=1}^N |\mu_{y_i}|$. This term monotonically decreases as the number of instances participating in the binary classification problem increases, so it encourages all $|\mu_k|$'s to be 1. Given a particular trade-off between these two components, the optimization aims to find two subsets of classes (while ignoring another subset) which can be separated as well as possible. For the constraints in (1), the first one defines the domain of μ_k, the second and third correspond to the hinge loss, the fourth is a balance constraint where the constant B is the tolerance of the violation of the balance constraint, and the last part requires that the coloring includes at least one positive class and one negative class.

Optimization. To solve (1), we use the alternating method to iteratively optimize over two sets of variables. Specifically, given some initial coloring $\{\mu_k^{(0)}\}$ (we will discuss how to initialize later), we fix $\{\mu_k\}$ and optimize over $\{\mathbf{w}, b, \{\xi_i\}\}$, and then fix $\{\mathbf{w}, b, \{\xi_i\}\}$ and optimize over $\{\mu_k\}$. This process is repeated until convergence or a prespecified number of steps.

With $\{\mu_k\}$ fixed, the resulting problem is a standard binary SVM on $S_x^+ \cup S_x^-$, and any suitable kernels can be used. With $\{\mathbf{w}, b, \{\xi_i\}\}$ fixed, we need to solve the following problem (grouping instances based on their classes):

$$
\begin{aligned}
\min_{\{\mu_k\}} \quad & C\sum_{k=1}^K n_k \big(\mathbf{1}\{\mu_k = +1\}\,(\frac{1}{n_k}\sum_{i:y_i=k} \xi_i^+ - \frac{A}{C}) \\
& \qquad + \mathbf{1}\{\mu_k = -1\}\,(\frac{1}{n_k}\sum_{i:y_i=k} \xi_i^- - \frac{A}{C})\big) \\
\text{s.t.} \quad & \mu_k \in \{-1, 0, +1\},\ \forall k \in \mathcal{Y} \\
& -B \leq \sum_{k=1}^K \mu_k \leq B \\
& \sum_{k=1}^K \mathbf{1}\{\mu_k > 0\} \geq 1 \text{ and } \sum_{k=1}^K \mathbf{1}\{\mu_k < 0\} \geq 1
\end{aligned} \tag{2}
$$

where $\xi_i^+ = \max\{0, 1-(\mathbf{w}^T\mathbf{x}_i+b)\}$ and $\xi_i^- = \max\{0, 1+(\mathbf{w}^T\mathbf{x}_i+b)\}$ correspond to the hinge loss when $\mathbf{x}_i$ is labeled as positive or negative respectively, and n_k is the number of instances from class k (for those k such that $\mu_k = \pm 1$,

$\sum_{i:y_i=k} |\mu_{y_i}| = n_k$).

There is a nice interpretation of the objective function of (2). To optimize over a single μ_k (ignore the balance constraint for now), we can just pick the value from $\{-1, 0, +1\}$ which gives the minimum of $\frac{1}{n_k}\sum_{i:y_i=k}\xi_i^- - \frac{A}{C}$, 0 and $\frac{1}{n_k}\sum_{i:y_i=k}\xi_i^+ - \frac{A}{C}$. Denote this minimum value for class k as μ'_k. We can see that whether a class k is active ($\mu'_k = \pm 1$) or ignored ($\mu'_k = 0$) depends on whether the average slack (hinge loss) $\frac{1}{n_k}\sum_{i:y_i=k}\xi_i^{-/+}$ is below or above the threshold $\rho \triangleq \frac{A}{C}$. Since the hinge loss is a truncated linear function of the margin, another interpretation is that if the average margin of a class is not sufficiently high, we will ignore it. *This criteria is one of the key differences between our method and existing methods, since the coloring depends directly on how far the instances are from the decision boundary instead of other metrics that are less correlated to the classification error.*

Given the balance constraint which couples different variables, it turns out that there exists an efficient algorithm with complexity $O(K \log K)$. Since $\{\mu'_k\}_{k=1}^K$ gives the lowest possible value of the objective function, if $-B \leq B' \leq B$ where $B' = \sum_{k=1}^K \mu'_k$, then $\{\mu'_k\}_{k=1}^K$ is the optimal solution of (2). If $B' > B$, starting from $\{\mu'_k\}_{k=1}^K$, to satisfy the constraint, we need to decrease the values of some μ_k's, *i.e.*, set $\mu_k = \mu'_k - 1$ or $\mu'_k - 2$. However, for any k, if the value of μ_k moves away from μ'_k, there will be a non-negative delta increase in the objective. Thus, intuitively the goal is to find a subset of μ_k's and decrease their values from μ'_k such that the balance constraint is satisfied while the delta increase in the objective is minimized. Basically, we can first compute the delta increase of the objective per unit decrease of μ_k for each k, sort them, and then select and decrease those μ_k's with the smallest increases of the objective until the balance constraint is satisfied. If $B' < -B$, we can do the analogous steps similar to the case of $B' > B$ by symmetry. Please refer to the supplementary materials [2] for the detailed discussion of the algorithm.

Initialization. In order to get good initialization for the alternating optimization procedure, we estimate an initial partition using two methods. The first one is a top-down approach similar to [15]. Specifically, we do normalized cut using the kernel matrix as the affinity matrix between pairs of instances. The coloring of the instances then votes the coloring of the classes. For example, if a majority of the instances from class k is colored by -1, then $\mu_k = -1$. However, we found that due to the strong performance of kernelized SVM, the iterative algorithm does not go far from the initial point. Therefore, we introduce a second bottom-up method. We first get a confusion matrix by training one-vs-all on the training set, and treat it as the affinity matrix between classes, where the higher the confusion, the stronger the affinity. We then sample multiple pairs of classes which have the least confusion. Each sampled pair of classes is used as an initial point (other classes are set to 0), and (1) is solved. Note that the confusion matrix could be estimated at different nodes only considering the remaining classes, but to reduce the training time, we only used a single global confusion matrix. We found that the bottom-up initialization gives better performance than the top-down method.

Model Selection for Nonlinear Kernels. If linear kernel is used, the optimization problem (1) well captures the factors related to the computational complexity, *i.e.*, the number of active classes and how balance the partition is. Thus, in this case, the model with the smallest objective value is considered the best. However, if nonlinear kernels are used, one important computational component, *i.e.* the number of support vectors, is missing in (1). Directly minimizing the number of support vectors seems to be very hard, so we designed another scoring function with the goal to better model the computational cost. We use this scoring function to score and select models from different initializations when solving (1). Given a fixed $\rho = \frac{A}{C}$, all of the models from different initializations will satisfy that the average hinge loss for active classes will be smaller than ρ. Therefore, we select the model with emphasis on minimizing the test time complexity. Given a node model $\mathcal{M} = (\{\mu_k\}, \mathbf{w}, b)$, let $N_{\text{SV}}(\mathcal{M})$ be the number of support vectors of the model $\mathcal{M}$. We define the *weighted average of the number of support vectors per class* for $\mathcal{M}$ as

$$s(\mathcal{M}) = p^-(\mathcal{M})\frac{N_{\text{SV}}(\mathcal{M})}{|S_y^+(\mathcal{M})|} + p^+(\mathcal{M})\frac{N_{\text{SV}}(\mathcal{M})}{|S_y^-(\mathcal{M})|} \qquad (3)$$

where $|S_y^+(\mathcal{M})|$ and $|S_y^-(\mathcal{M})|$ are the number of positive and negative classes respectively, and $p^-(\mathcal{M}) = \frac{|S_y^-(\mathcal{M})|}{|S_y^-(\mathcal{M})|+|S_y^+(\mathcal{M})|}$ is the percentage of the negative classes, and similarly $p^+(\mathcal{M}) = 1 - p^-(\mathcal{M})$ is the percentage of the positive classes. (3) reflects the average efficiency of model $\mathcal{M}$. Consider an instance of class k from S_y^-. Ideally, after evaluating $\mathcal{M}$, a set of classes $S_y^+(\mathcal{M})$ can be pruned with the cost of N_{SV} kernel evaluations. Thus, the average cost of eliminating one class is $\frac{N_{\text{SV}}(\mathcal{M})}{|S_y^+(\mathcal{M})|}$. Similarly, for instances from $S_y^+(\mathcal{M})$, the average cost of eliminating one class is $\frac{N_{\text{SV}}(\mathcal{M})}{|S_y^-(\mathcal{M})|}$. Given the proportions of the positive $p^+(\mathcal{M})$ and negative classes $p^-(\mathcal{M})$, the weighted average of the cost of eliminating one class by $\mathcal{M}$ is calculated as (3). We select the model with the smallest $s(\mathcal{M})$.

Hierarchy Structure. A whole spectrum of hierarchies can be generated by varying $\rho = \frac{A}{C}$. Specifically, by setting ρ to be large, we encourage all classes to participate at a node resulting in a hierarchical tree structure similar to [4, 13, 14, 25, 10]. In this case, the model may suffer

[2] www.stanford.edu/~tianshig/papers/learningRelaxedHierarchy-ICCV11-supp.pdf

from high classification error, since the average hinge loss at each node is likely to be high due to not ignoring confusing classes. However, the test speed is likely to be fast, since the length of the decision path is short. By setting ρ to be small, only the two classes that can be separated best are used in a node. This gives rise to a DAGSVM model [17] with a learned decoding order. In this case, although the average hinge loss is lower, only a few classes are considered in a node, so the decision path is longer, resulting in slower test time. Furthermore, the number of nodes in the model will be larger, which implies a higher model complexity, potentially leading to lower generalization error (see Section 4). One can set ρ according to the application-specific trade-off between the accuracy and speed.

Another observation is that as the hierarchy grows down, it gets harder and harder to find two subsets of classes with average hinge loss below ρ due to stronger similarity between the remaining classes. One can increase ρ to encourage more classes to participate, but this leads to inferior accuracy due to mixing of many similar classes. To get good accuracy, one can keep or even decrease ρ such that each node only considers two classes with good separability. This reduces to the case of DAGSVM for lower levels in the hierarchy. Another way used by [10, 15] is that, when it's hard to partition the classes in a node, a one-vs-all classifier is trained for this node without further going down. We found that both keeping (or decreasing) ρ and training one-vs-all for lower level nodes give similar complexity, but the accuracy is slightly better for the latter. Therefore, we used the latter method in our experiment.

4. Analysis of Generalization Error

We provide an analysis of the generalization error in the supplementary materials. The main insights from the theoretical analysis are that the larger the margins in the nodes and/or the fewer number of nodes, the lower the error bound. Our method seeks partitions such that the margins can be maximized. We also encourage more active classes and balanced partitions not only for computational efficiency but also for lower error bound (fewer decision nodes). Usually margins can be increased by having fewer active classes resulting in more decision nodes, and vice versa, so a good balance should be established by ρ.

5. Experiments

Basic Setup. We experiment on two fundamental computer vision tasks: object recognition and scene classification. We first describe the evaluation metrics in terms of classification accuracy and test speed. For the classification accuracy, we use the mean of per-class accuracy, which is the standard way to report multiclass classification performance [9]. For the test speed, we distinguish two cases. The first case corresponds to linear classifiers. In this case, the complexity of each classifier is the same, so the overall test complexity is proportional to the number of evaluated classifiers. Therefore, when linear kernel is used for SVM, we report the mean of the number of classifier evaluations for all test instances. The second case corresponds to nonlinear classifiers. Here, we focus on SVM with nonlinear kernels. In this case, the complexity of each classifier is no longer the same but is proportional to the number of its support vectors. Specifically, let M be the number of classifiers to be evaluated. Each of them has a set of support vectors $\{SV_i^{(m)}\}$ where m and i index the classifier and its support vectors respectively. One way to do evaluation is to evaluate M classifiers independently without caching any kernel computations. This results in a number of $\sum_{m=1}^{M} |\{SV_i^{(m)}\}|$ kernel computations for a single test instance. Another way is to cache the kernel computations from different binary classifiers and reuse them whenever possible. In this way, the number of kernel evaluations can be reduced to $|\cup_{m=1}^{M} \{SV_i^{(m)}\}|$. The first way is very inefficient when there are a large number of classifiers but with shared support vectors (trained on the common training set), which is the case of large-scale multiclass classification. Therefore, we use the second way to report the test speed when nonlinear kernels are used.

We compare our method to various existing approaches: one-vs-all, one-vs-one, DAGSVM, tree-based hierarchy [10] and Marszalek [15]. Note that Marszalek [15] has been shown to be much better than earlier standard hierarchical methods, *e.g.*, [4, 13, 14, 25]. For the tree-based hierarchy method [10], we used the top-down method for building the taxonomies and did not perform subsampling for the branch nodes as [10], since it is an orthogonal design that can be applied to any hierarchy based methods. All methods use SVM as the binary classifier. The parameter C of SVM is chosen by cross validation on the training set. For the bottom-up initialization of our method, we used 30 samples. We set $B = 10$ in (1) for all experiments.

Caltech-256. We first experimented on the object recognition task on Caltech-256 [9]. The Caltech-256 dataset is a standard multiclass object recognition dataset with 256 categories and at least 80 images per class. We randomly sampled 80 images for each class, and split them into one half for training (40 per class) and the other half for test (40 per class). For the feature, we used the standard spatial histograms of visual words based on dense SIFT. We used the same extended Gaussian kernel based on χ^2 distance as [15]. We also tested using the linear kernel, since the computational complexity is measured by the number of classifier evaluations instead of the number of kernel evaluations using nonlinear kernels. However, since the linear kernel on the histogram based feature gives very poor accuracy, we used the explicit feature transformation from [21] to approximate the implicit fea-

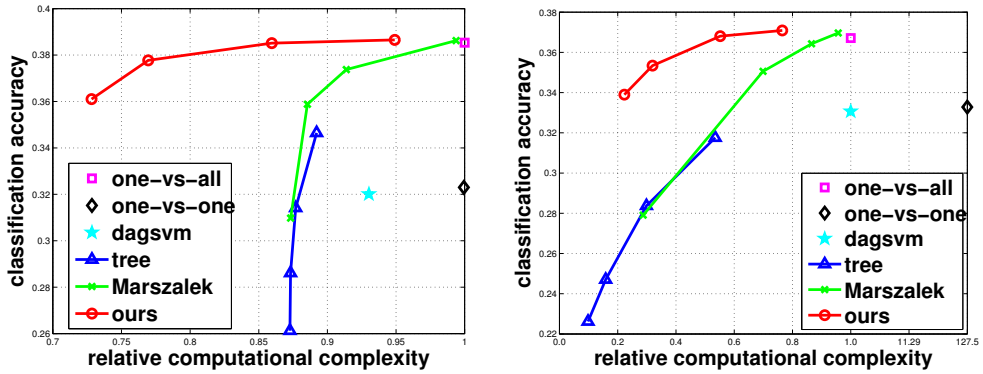

Figure 2. Performance comparison on Caltech-256. The left is the results using extended Gaussian kernel based on χ^2 distance, and the right is using linear kernel. The computational complexity is normalized by the complexity of one-vs-all. Note that for one-vs-one with linear kernel, the relative complexity is 127.5.

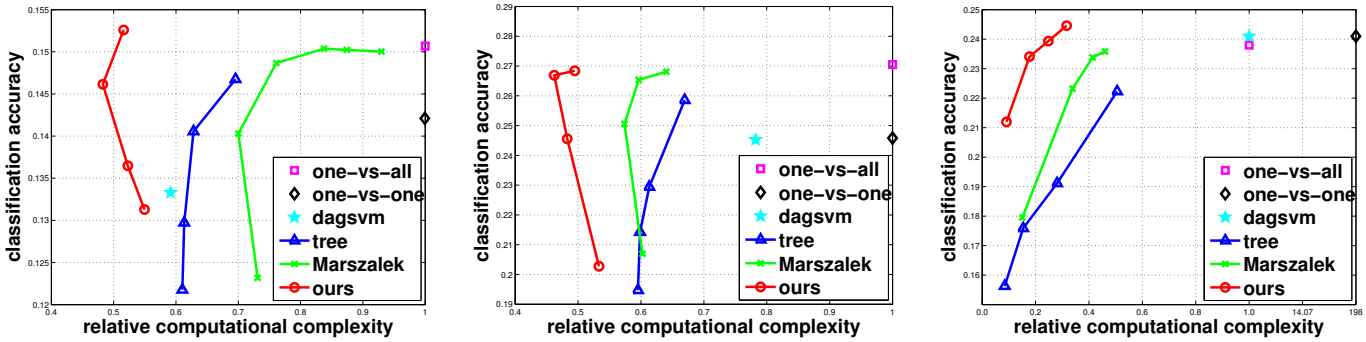

Figure 3. Performance comparison on the SUN dataset. From left to right: GIST feature with RBF kernel, spatial HOG feature with histogram intersection kernel, and transformed spatial HOG feature with linear kernel. The computational complexity is normalized by the complexity of one-vs-all. Note that for one-vs-one with linear kernel, the relative complexity is 198.

ture mapping of χ^2 kernel. We then apply linear SVM on the transformed feature. We varied the trade-off between accuracy and speed for tree [10] (2 to 5 levels), Marszalek [15] ($\alpha \in \{0.2,\ 0.5,\ 0.6,\ 0.8\}$) and our method ($\rho \in \{0.5,\ 0.6,\ 0.7,\ 0.8\}$).

The results are shown in Figure 2. As can be seen, for both cases of nonlinear and linear kernels, our method has much better performance than all the other methods (faster at the same accuracy level, more accurate at the same complexity). For example, in the case of linear kernel, without any sacrifice of the accuracy (in fact ours achieves one of the best accuracy), our method has only around 50% of the complexity of one-vs-all. With around 3% accuracy decrease, our method is 5x faster than one-vs-all. However, to achieve the same level of speedup, Marszalek [15] has to suffer from around 8% accuracy decrease, and tree-based hierarchy [10] has 10% accuracy decrease.

SUN-397. We then experimented on the scene classification task on the SUN dataset [24]. The SUN dataset is by far the largest scene recognition dataset. It captures a full variety of 899 scene categories. We used 397 well-sampled categories to run the experiment as [24]. For each class, 50 images are used for training and the other 50 for test. For image representation, we changed from SIFT based feature to another three different feature and kernel pairs to test whether the performance gain of our method is consistent with different representations. The first pair is GIST [16] with RBF kernel. The second is spatial HOG pyramid [24] with histogram intersection kernel. The last is transformed spatial HOG pyramid (the explicit feature transformation from [21] to approximate the implicit feature mapping of histogram intersection kernel) with linear kernel. We varied the trade-off between accuracy and speed for tree [10] (2 to 5 levels), Marszalek [15] ($\alpha \in \{0.1,\ 0.2,\ 0.3,\ 0.4,\ 0.6,\ 0.8\}$ or a subset for HOG) and our method (ρ from 0.6 to 0.9 with step size 0.05).

The results are shown in Figure 3. The performance boost for three different cases is consistent and similar to that on the Caltech dataset. For example, for GIST with RBF kernel, with slightly better accuracy, our method has 51.6% complexity compared to one-vs-all. However, Marszalek [15] can only reduce the relative complexity to 76% (with similar accuracy as one-vs-all) and tree-based hierarchy [10] has relative complexity 70% but with more accuracy decrease. Similarly, when HOG with histogram intersection kernel is used, with the same accuracy, our method is 2x faster than one-vs-all, and the other two methods never met this speedup no matter which accuracy level

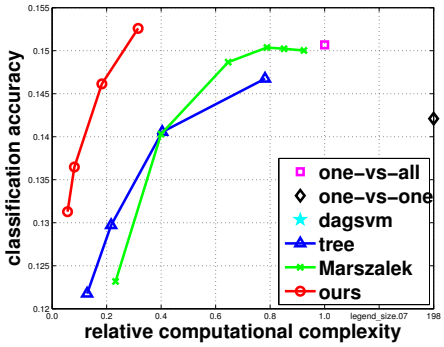

Figure 4. Illustration of using the number of classifier evaluations as complexity metric when nonlinear kernel is used (GIST with RBF kernel on SUN dataset).

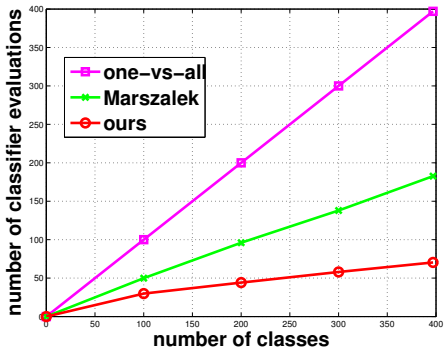

Figure 5. Complexity over the number of classes.

is. In addition, for HOG with linear kernel, our method can achieve the same accuracy (or slightly better) as one-vs-all and DAGSVM, while being 4x faster. Furthermore, with around 2.6% decrease in accuracy, our method is 11x faster than one-vs-all and DAGSVM. However, for the other two hierarchical methods, to achieve the same level of speedup, their accuracy decreased by around 7%.

It is worth noting that if nonlinear kernels are used, a lower height of the hierarchy does not necessarily lead to lower computational complexity. This can be seen from the first two plots in Figure 3. When α is large for [15] or ρ is large for our method, the height of the hierarchy is low but both the accuracy and the complexity get worse. The reason is that, although a fewer number of classifier evaluations are needed, each classifier involves a large number of support vectors (since it's trained on instances from a large subset of classes), which makes the overall complexity high. This also implies that full tree based methods [22, 4, 13, 14, 25, 3] may have much less computational gain compared to the linear classifier case when nonlinear kernels are used. To emphasize the importance of using the correct complexity metric ([15] used nonlinear SVMs but the complexity is measured using the number of classifier evaluations), Figure 4 shows an example of using the number of classifier evaluations as complexity metric when nonlinear kernel is used. The correct picture is the first plot in Figure 3, which is very different from Figure 4. The complexity is heavily underestimated for the hierarchical methods in Figure 4. In addition, the complexity of one-vs-one is heavily overestimated, since the first plot in Figure 3 shows that one-vs-one has the same complexity as one-vs-all, but in terms of the number of classifier evaluations, one-vs-one is 198 times slower than one-vs-all.

We also studied how the complexity changes over number of classes. We randomly sampled 100, 200 and 300 classes together with the original 397 classes from the SUN dataset, and for each case, we learn the model with spatial HOG using linear kernel. In addition, for each case except for 397 classes, the experiment is repeated 5 times, and the mean is reported. To be fair, we set $\rho = 0.7$ for our method, and $\alpha = 0.4$ for [15] to match the same level of accuracy as one-vs-all. As shown in Figure 5, the complexity of our method grows sublinearly and more slowly than that of [15].

A hierarchy is built in the label space automatically by our method. We present a sampling of the first three levels of the hierarchy (Figure 6) learned based on the spatial HOG feature with histogram intersection kernel. We show a more complete version and the learned hierarchy on Coaltech-256 in the supplementary materials. There are several interesting aspects of the learned hierarchy: first, in many cases the coloring of the positive and negative classes correlates with human vision, *i.e.*, images from different classes but within the same partition look similar to human and vice versa; second, the coloring of the classes for the first two levels correlates with human defined concepts, *e.g.*, natural outdoor scenes v.s. indoor man-made scenes; last, the hierarchy starts from partitioning classes with large visual distances and then distinguishes more subtle differences at lower level.

6. Conclusion and Future Work

In this paper, we proposed a method for discriminative learning of relaxed hierarchy by jointly learning the coloring of classes and the induced binary classifier in a unified and principled max-margin optimization. Our method has been tested on both the Caltech-256 (object recognition) and SUN (scene classification) datasets, and shows significant improvement over existing methods consistently with various representations. One future direction is to learn the hierarchy in a less greedy and more systematic way. For example, one can look ahead for multiple levels, non-uniformly weight instances to encourage support vectors from higher levels to be reused in lower levels and so on. Another direction is to consider multiple different features in a single hierarchy. Instead of pre-computing all types of features at once, we can dynamically choose which feature is best suitable for discriminating the remaining classes at a given level. More generally, one can view the hierarchy as a guidance to select the most suitable models/features from a hybrid set for the problem at a certain node or level.

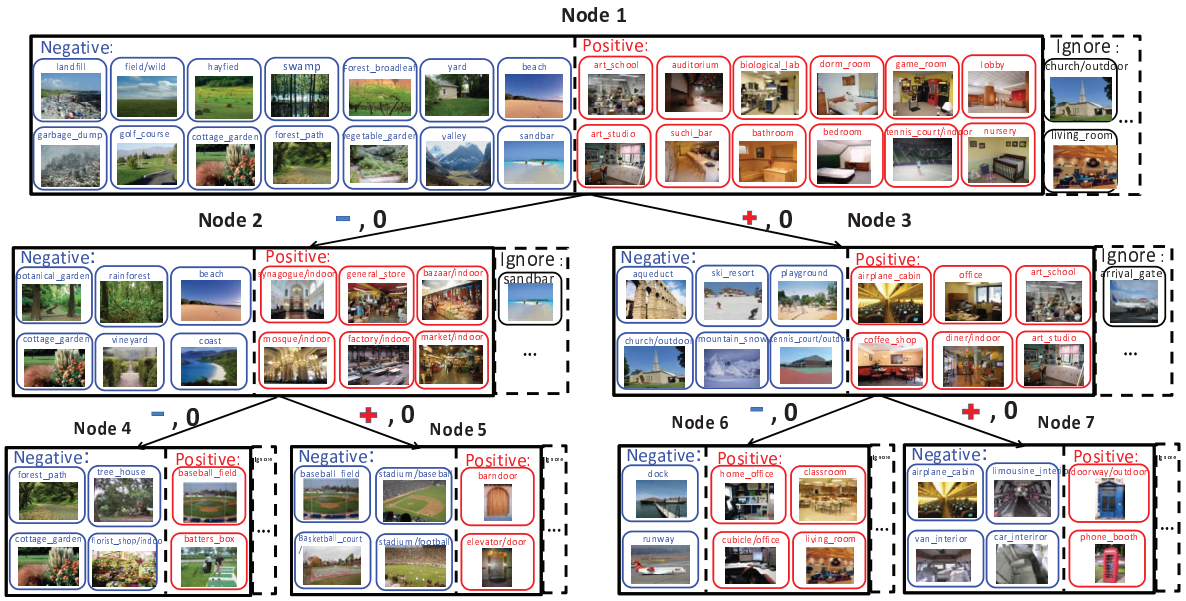

Figure 6. A sampling of the learned hierarchy using HOG on the SUN dataset. Please see more in the supplementary material.

Acknowledgment. This work was supported by the NSF under grant No. RI-0917151 and the Office of Naval Research MURI grant N00014-10-10933.

References

[1] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *J. Mach. Learn. Res.*, 2001. 1

[2] E. Bart, I. Porteous, P. Perona, and M. Welling. Unsupervised learning of visual taxonomies. In *CVPR*, 2008. 2

[3] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multiclass task. In *NIPS*, 2010. 2, 7

[4] Y. Chen, M. Crawford, and J. Ghosh. Integrating support vector machines in a hierarchical output space decomposition framework. In *IGARSS*, 2004. 2, 4, 5, 7

[5] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2002. 2

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009. 1

[7] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *J. A.I. Res.*, 95. 1

[8] S. Escalera, M. Tax, O. Pujol, P. Radeva, and D. R. P.W. Subclass problem-dependent design for error-correcting output codes. *PAMI*, 30, 2008. 2

[9] G. Griffin, A. Holub, and P. P. Caltech-256 object category dataset. Technical report, Cal. Tech., 2007. 1, 5

[10] G. Griffin and P. Perona. Learning and using taxonomies for fast visual categorization. In *CVPR*, 2008. 2, 4, 5, 6

[11] J. Li, C. Wang, Y. Lim, D. Blei, and L. Fei-Fei. Building and using a semantivisual image hierarchy. In *CVPR*, 2010. 2

[12] L. Li. Multiclass boosting with repartitioning. In *ICML06*. 1

[13] S. Liu, H. Yi, L. Chia, and R. Deepu. Adaptive hierarchical multi-class svm classifier for texture-based image classification. In *ICME*, 2005. 2, 4, 5, 7

[14] Z. Liu, W. Shi, Q. Qin, L. X., and X. D. Hierarchical support vector machines. In *IGARSS*, 2005. 2, 4, 5, 7

[15] M. Marszalek and S. C. Constructing category hierarchies for visual recognition. In *ECCV*, 2008. 1, 2, 4, 5, 6, 7

[16] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 2001. 6

[17] J. C. Platt, N. Cristianini, and J. Shawe-taylor. Large margin dags for multiclass classification. In *NIPS*, 2000. 1, 5

[18] O. Pujol, P. Radeva, and J. Vitria. Discriminant ecoc: A heuristic method for application dependent design of error correcting output codes. *PAMI*, 28:1007–1012, 2006. 2

[19] R. E. Schapire. Using output codes to boost multiclass learing problems. In *ICML*, 1997. 1

[20] J. Sivic, B. Russell, A. Zisserman, W. Freeman, and A. Efros. Unsupervised discovery of visual object class hierarchies. In *CVPR*, 2008. 2

[21] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. In *CVPR*, 2010. 5, 6

[22] V. Vural and J. G. Dy. A hierarchical method for multi-class support vector machines. In *ICML*, 2004. 2, 7

[23] S. Xia, J. Li, L. Xia, and C. Ju. Tree-structured support vector machines for multi-class classification. In *ISNN*, 2007. 2

[24] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010. 1, 6

[25] X. Yuan, W. Lai, T. Mei, X. Hua, X. Wu, and S. Li. Automatic video genre categorization using hierarchical svm. In *ICIP*, 2006. 2, 4, 5, 7

[26] A. Zweig and D. Weinshall. Exploiting object hierarchy: Combining models from different category levels. In *ICCV*, 2007. 2