# Hierarchical Multiclass Object Classification

Fereshte Khani
Massachusetts Institute of Technology
fereshte@mit.edu

Anurag Mukkara
Massachusetts Institute of Technology
anurag_m@mit.edu

## Abstract

*Human can use similarity between objects in order to recognize rare objects. They also make many abstract concepts when they see some objects very often. Interestingly, a large part of brain is associated with common classes like faces rather than rare objects like Ostrich. In our work we want to propose a model that has four mentioned characteristics. 1. Use more resources for categories that have many examples and less resources for categories that have few examples. 2. Has the ability recognize objects that it has never seen before, but can be made by combining previously seen objects. 3. Objects with few examples can borrow statistical strength from similar objects with many examples. 4. Models that have lot of sharing between model parameters of different classes should be favored.*

## 1. Introduction

## 2. Previous work

## 3. Hierarchical Classification Model

### 3.1. Learning Independent Classification Models

Consider a classification problem where we observe a dataset $D$ of labelled training examples. Each example belongs belongs to one $K$ classes(eg. 40 object categories). The dataset $D$ has $n_k$ labeled examples for for each class $k \in 1, 2, ...., K$. We $x_i^{(k)} \in R^D$ be the input feature vector of length D for the $i^{th}$ training example of class $k$ and $y_i^{(k)}$ be the corresponding class label. $y_i^k = 1$ indicates that the $i^{th}$ training example is a positive example for class $K$ and $y_i^k = -1$ indicates that the $i^{th}$ training example is a negative example for class $K$.

For a binary classification problem we can use a simple logistic regression model. In particular, for each class $k$, the probability of a positive instance can be modeled as:

$$p(y_i^{(k)} = 1|\beta^{(k)}) = \frac{\exp(\beta^{(k)^T} x_i^{(k)})}{1 + \exp(\beta^{(k)^T} x_i^{(k)})} \tag{1}$$

where $k$ ranges over the $K$ classes and $\beta^k \in R^D$ is the unknown model parameter of length D for class k. In addition, we place a zero-mean spherical Gaussian prior over each model parameter $\beta^k$:

$$p(\beta) = \prod_{k=1}^{K} p(\beta^{(k)}) = \prod_{k=1}^{K} N(0, \frac{1}{\lambda}) \tag{2}$$

where $N(\mu, \Sigma)$ indicates a Gaussian probability distribution with mean $\mu$ and covariance $\Sigma$. Here, $\lambda$ is the prior parameter. With this modeling framework, the log posterior distribution of the unknown parameters can be expresses as

$$\log p(\beta|D) \propto \sum_{k=1}^{K} \Big[ \sum_{i=1}^{n_k} \log p(y_i^{(k)}|x_i^{(k)}, \beta^{(k)}) + \log p(\beta^{(k)}) \Big] \tag{3}$$

Maximizing the log posterior(or finding the MAP estimate ) over the unknown model parameters $\beta$ is equivalent to minimizing the probabilistic loss function with quadratic regularization terms:

$$E = \sum_{k=1}^{K} \Big[ \sum_{i=1}^{n_k} Loss(y_i^{(k)}, x_i^{(k)}, \beta^{(k)}) + \frac{\lambda}{2} \|\beta^{(k)})\|^2 \Big] \tag{4}$$

where the probabilistic loss function is given by:

$$Loss(y, x, \beta) = - \sum_{j \in -1,1} I\{y = j\} \log p(y = j|x, \beta) \tag{5}$$

One important property of the loss function is that it is convex and hence can be efficiently optimized using standard convex optimization solvers.

The main drawback of this formulation is that all the $K$ classes are treated as unrelated entities and hence do not share any model parameters. In fact, it is observed that the objective function decomposes into $K$ sub-problems, each of which is a binary classification problem that can be trained independently using the training data for that particular class.

## 3.2. Learning a Hierarchical Classification Model

In our framework, a hierarchical tree structure is used to model sharing of parameters between visually related classes. Each node in the tree has a model parameter, a vector of length $D$ associated with it. Let $\theta = \{\theta_1, \theta_2, ....., \theta_n\}$ denote the model parameters of all the nodes in the tree, where $n$ is the number of nodes starting from the root node to the all the leaf nodes. The leaf nodes correspond to the $K$ object categories or classes. The parameters for a particular class $k$ is the sum of the parameters of all the nodes, say $i_1, i_2, ..., i_l$, along the path from the leaf node corresponding to that class up until the root node of the tree:

$$\beta^{(k)} = \theta_{i_1} + \theta_{i_2} + ...... + \theta_{i_l} \tag{6}$$

We place zero-mean spherical Gaussian priors on the parameters of each node. However, the covariance matrix is dependent on the level of a particular node in the tree i.e all the nodes at a particular level have the same prior parameters.

$$p(\theta_i) = N(0, \frac{1}{\lambda_l} I) \tag{7}$$

where node $i$ is at level $l$ in the tree. The root node has level 0 and the level increases as we move towards the leaf nodes.

With this modification in the way the model parameters for classes are related, the hierarchical learning problem can be formulated as minimizing the log posterior or equivalently the loss function with respect to $\beta$:

$$\begin{aligned} E = Loss(Y, X, \beta) + \frac{\lambda_0}{2}\|\theta_0\|^2 + \sum_{i=1}^{k_1} \frac{\lambda_1}{2}\|\theta_{i1}\|^2 \\ + \sum_{i=1}^{k_2} \frac{\lambda_2}{2}\|\theta_{i2}\|^2 + ...... + \sum_{i=1}^{k_l} \frac{\lambda_l}{2}\|\theta_{il}\|^2 \end{aligned} \tag{8}$$

where $Loss(Y, X, \beta)$ is the loss function as explained in the previous section but with the additional constraint that $\beta$ satisfies equation 6 according to the tree structure. Here there are $k_j$ nodes in level $j$ and $\theta_{1j}, \theta_{2j}, ....., \theta_{k_j j}$ are the parameters of nodes in level $j$.

## 3.3. Learning the tree structure

If we are given the tree structure that indicates the relation between different classes, then the learning algorithm only needs to optimize for the loss in equation 8. However, as discussed in [?], learning the tree structure dynamically based on the input classes and training data could lead to better results in object classification. We adopt the algorithm used in [?] to multilevel trees and fine-grained partitioning. In [?], the author use a fixed number of levels i.e 3 in the tree and use a non-parametric Chinese Restaurant Process Prior(CRP) to model the position of a class in the tree. Although this simple CRP prior allows one to have unknown and potentially unbounded number of groups of object classes, this is can only be used for two-level trees. For trees which have a fixed number of levels more than 3, one could use a nested CRP prior [?]. Since this approach has the drawback of limiting the tree to a fixed number of levels, we use a simple extension(modified-CRP) of the CRP prior used in [?] to allow the flexibility of having different depths at different parts of the tree.

A new class could go into one of the existing super-categories or create a new super-category. A new super-category can be attached to any node which doesn't have any children that are leaf nodes. That is, we allow the parent of a new super-category to be any node that is not itself a super-category that has leaf nodes, which correspond to classes, as children. By following the procedure in [?], for each possible position of the new class we calculate a likelihood term and modified-CRP prior term. The position of the new class will be the one that maximizes the sum of likelihood term and prior term.

Let $i$ denote the node number of the leaf node corresponding to the new class. To calculate the likelihood at a possible position in the tree, we find the model parameter $\theta_i$ of the leaf node, that maximizes the log likelihood of the training data at that position. This likelihood will depend on the position in the tree, when we choose a position it means that we include the parameters of all the nodes that are ancestors of the leaf node $i$ at that position. If the position corresponds to a category that is visually dissimilar to the new class, then the best we can with $\theta_i$ with still result in a lower likelihood term. If the the category is visually similar to the new class, we can achieve a far better fit and the likelihood term goes up. We have observed that the maximum of the likelihood over the model parameter $\theta_i$ is greatly affected by the position we choose for the leaf node. In addition to the likelihood term, the modified-CRP prior term favors super-categories that have a large number of leaf nodes or class under them. The modified-CRP prior is given by:

$$p(z = j) = \begin{cases} \frac{m_j}{n+\gamma * m} & : j \text{ is old} \\ \frac{\gamma}{n+\gamma * m} & : j \text{ is new} \end{cases}$$

where $n$ is the total number of leaf nodes, $m_j$ is the number of leaf nodes under super-category $j$, $\gamma$ is the concentration parameter that controls the probability of creating a new super-category, $m$ is the total number of positions of new super-categories in the tree. $m$ is equal to number of nodes in the tree that have only super-categories as children. In other words, $m$ is the cardinality of the set that includes the root node as well all nodes that are not leaf nodes or parents of leaf nodes. It can be easily verified that this defines a valid probability distribution over the choice $z$ of possible positions in the tree where the new class can be placed.

We use same model-fitting algorithm described in [**?**] with modifications to CRP-prior term and tree structure as described above. Basically, the algorithm alternates between deciding the position in the tree of a new class and optimizing the model parameters of all the nodes in the tree for a given tree structure. This alternating procedure will reach a local minima of the loss function described in equation 8. Since the loss function is convex, this is also the global minima of the loss function. For optimizing the model parameters for a fixed tree structure , we use iterative coordinate-descend procedure. Since the parameters of all the nodes at any given level of the tree are independent of each other, we parallelize the loss minimization routine for these nodes. This leads to significant reduction in time taken to learn the model.

We also use a simple heuristic to maintain a fine-partitioning of the classes among into among super-categories. We need to avoid many leaf nodes accumulating under the same super-category as this would lead to a very coarse-grain partitioning of the classes. So, whenever the number of leaf nodes under a super-category goes above a certain threshold $maxChild$, we break down the subtree under that node to create a more fine-grained partitioning. To do this we run the entire optimization algorithm described above on the classes that were originally under this super-category. Basically, we consider the node corresponding to the super-category as a root node and keep adding new leaf nodes to form a tree under this root node. In most cases, the $maxChild$ number of classes that previously shared the same parent are dividing several finer super-categories and have different parents. In the rare case when the classes are very similar to each other, this process will add redundant to the tree. We identify this pathology and remove the redundant node while appropriately modifying the modle parameters of the relevant nodes. The parameter $maxChild$ can be configured to control the degree of fine-grained partitioning of classes. A lower value of $maxChild$ leads to very fine-grained partitions whereas a higher value of $maxChild$ results in coarser partitions. We found that a value of four results in best results.