Probabilistic Abstraction Hierarchies

Eran Segal
Computer Science Dept.
Stanford University
eran@cs.stanford.edu

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

Dirk Ormoneit
Computer Science Dept.
Stanford University
ormoneit@cs.stanford.edu

Abstract

Many domains are naturally organized in an abstraction hierarchy or taxonomy, where the instances in "nearby" classes in the taxonomy are similar. In this paper, we provide a general probabilistic framework for clustering data into a set of classes organized as a taxonomy, where each class is associated with a probabilistic model from which the data was generated. The clustering algorithm simultaneously optimizes three things: the assignment of data instances to clusters, the models associated with the clusters, and the structure of the abstraction hierarchy. A unique feature of our approach is that it utilizes global optimization algorithms for both of the last two steps, reducing the sensitivity to noise and the propensity to local maxima that are characteristic of algorithms that only take local steps. We provide a theoretical analysis for our algorithm, showing that it converges to a local maximum of the probability of model and data. We present experimental results on synthetic data, and on real data in the domains of gene expression and text.

1 Introduction

Many domains are naturally associated with a hierarchical taxonomy, in the form of a tree, where instances that are close to each other in the tree are assumed to be more "similar" than instances that are further away. In biological systems, for example, creating a taxonomy of the instances is one of the first steps in understanding the system. Thus, in particular, much of the work on analyzing gene expression data [3] has focused on creating gene hierarchies. Similarly, in text domains, creating a hierarchy of documents is a common task [11].

In many of these applications, the hierarchy is unknown; indeed, discovering the hierarchy is often a key part of the analysis. The standard algorithms applied to the problem typically use an agglomerative bottom-up approach [3] or a divide-and-conquer top-down approach [7]. Although these methods have been shown to be useful in practice, they suffer from several limitations: First, they proceed via a series of local improvements, making them particularly prone to local maxima. Second, at least the bottom-up approaches are typically applied to the raw data; models (if any), are constructed as a post-processing step. Thus, domain knowledge about the type of distribution from which data instances are sampled is rarely used in the formation of the hierarchy.

In this paper, we present *probabilistic abstraction hierarchies (PAH)*, a probabilistically principled general framework for learning abstraction hierarchies from data which overcomes these difficulties. We use a Bayesian approach, where the different models correspond to different abstraction hierarchies. The prior is designed to enforce our intuitions about taxonomies: nearby classes have similar data distributions. More specifically, a model in a PAH is a tree, where each node in the tree is associated with a class-specific probabilistic model (CPM). Data is generated only at the leaves of the tree, so that a model basically defines a mixture distribution whose components are the CPMs at the leaves of the tree. The CPMs at the internal nodes are used to define the prior over models: We prefer models where the CPM at a child node is close to the CPM at its parent, relative to some distance function between CPMs. Our framework allows a wide range of notions of distance between models; we essentially require only that the distance function be convex in the parameters of the two CPMs. For example, if a CPM is a Gaussian distribution, we might use a simple squared Euclidean distance between the parameters of the two CPMs.

We present a novel algorithm for learning the model parameters and the tree structure in this framework. Our algorithm is based on the structural EM (SEM) approach of [4], but utilizes "global" optimization steps for learning the best possible hierarchy and CPM parameters (see also [5, 12] for similar global optimization steps within SEM). Each step in our procedure is guaranteed to increase the joint probability of model and data, and hence (like SEM) our procedure is guaranteed to converge to a local optimum.

Our approach has several advantages. (1) It provides principled probabilistic semantics for hierarchical models. (2) It is model based, which allows us to exploit domain structural knowledge more easily. (3) It utilizes global optimization steps, which tend to avoid local maxima and help make the model less sensitive to noise. (4) The abstraction hierarchy tends to pull the parameters of one model closer to those of nearby ones, which naturally leads to a form of parameter smoothing or *shrinkage* [11].

We present experiments for PAH on synthetic data and on two real data sets: gene expression and text. Our results show that the PAH approach produces hierarchies that are more robust to noise in the data, and that the learned hierarchies generalize better to test data than those produced by hierarchical agglomerative clustering.

2 Probabilistic Abstraction Hierarchy

Let S be the domain of some random observation, e.g., the set of possible assignments to a set of features. Our goal is to take a set of instances in S, and to cluster them into some set of k classes. Standard "flat" clustering approaches — for example, Autoclass [1] or the k-means algorithm — are special cases of a generative mixture model. In such models, each data instance belongs to one of the k classes, each of which is associated with a different *class-specific probabilistic model* (CPM). Each data instance is sampled independently by first selecting one of the k classes according to a multinomial distribution, and then randomly selecting the data instance itself from the CPM of the chosen class.

In standard clustering models, there is no relation between the individual CPMs, which can be arbitrarily different. In this paper, we propose a model where the different classes are related to each other via an abstraction hierarchy, such that classes that are "nearby" in the hierarchy have similar probabilistic models. More precisely, we define:

Definition 2.1 *A* probabilistic abstraction hierarchy (PAH) $\mathcal{A}$ *is a tree* T *with nodes* $V = \{v_1, \ldots, v_m\}$ *and edges* E*, such that* T *has exactly* k *leaves* $v_1, \ldots, v_k$*. Each node* v_i*,* $i = 1, \ldots, m$*, is associated with a CPM* M_i*, which defines a distribution over* S*; we use* $\boldsymbol{M}$ *to denote* $M_1, \ldots, M_m$*. We also have a multinomial distribution over the leaves* $v_1, \ldots, v_k$*; we use* $\boldsymbol{\theta}$ *to define the parameters of this distribution.*

We note that the tree in a PAH can use directed or undirected edges. For simplicity, we focus attention on undirected trees.

Our framework does not, in principle, place restrictions on the form of the CPMs; we can use any probabilistic model that defines a probability distribution over S. For example, M_i may be a Bayesian network, in which case its specification would include the parameters, and perhaps also the network structure; in a different setting, M_i may be a hidden Markov model. In practice, however, the choice of CPMs has ramifications both for the overall hierarchical model and the algorithm.

As discussed above, we assume that data is generated only from the leaves of the tree; the internal nodes serve to define the hierarchy, enforcing the intuition that CPMs that are nearby in the tree are similar to each other. Thus, we augment S with an additional hidden class variable C for each data item, which takes the values $1, \dots, k$ depending on the leaf that was chosen to generate this item. Given a PAH $\mathcal{A}$, an element $s \in S$, and a value c for C, we define $P(s, c \mid \mathcal{A}) = P(C = c \mid \boldsymbol{\theta})P(s \mid M_c)$, where $P(C = c \mid \boldsymbol{\theta})$ is the multinomial distribution over the leaves and $P(s \mid M_c)$ is the conditional density of the data item given the CPM at leaf c. The conditional distribution of s given $\mathcal{A}$, from which the data are generated, is simply $P(s \mid \mathcal{A})$, where c is summed out from $P(s, c \mid \mathcal{A})$.

As we mentioned, the role of the abstraction hierarchy is to define an intuitive ordering of the CPMs by enforcing similarity between the CPMs at nearby leaves. We achieve this goal by defining a prior distribution over abstraction hierarchies $\mathcal{A}$ that penalizes the distance between neighboring CPMs M and M' using a distance function $\rho(M, M')$. Note that we do not require that ρ be a distance in the mathematical sense; instead, we only require that it be symmetric (as we chose to use undirected trees), non-negative, and that $\rho(M, M') = 0$ iff $M = M'$.[1] For example, if M and M' have the same parameterization, we might choose $\rho(M, M')$ to be the sum of squared distances between the parameters of M and M'. Another choice is to define $\rho(M, M') = \boldsymbol{D}_{KL}(M; M') + \boldsymbol{D}_{KL}(M'; M)$, where $\boldsymbol{D}_{KL}(M; M')$ is the KL-distance between the distributions that M and M' define over S. This distance measure has the advantage of being applicable to any pair of CPMs over the same space, even if their parameterization is different. Given a definition of ρ, we define the prior over PAHs as

$$P(\mathcal{A}) \propto \prod_{(i,j) \in E} \exp\left(-\lambda \rho(M_i, M_j)\right),$$

where λ represents the extent to which differences in distances are penalized (larger λ represents a larger penalty).[2]

Given a set of data instances D with domain S, our goal is to find a PAH $\mathcal{A}$ that maximizes $P(\mathcal{A} \mid D) \propto P(\mathcal{A})P(D \mid \mathcal{A})$ or equivalently, $\log P(\mathcal{A}) + \log P(D \mid \mathcal{A})$. By maximizing this expression, we are trading off the fit of the mixture model over the leaves to the data D, and the desire to generate a hierarchy in which nearby models are similar.

3 Learning the Models

Our goal in this section is to learn a PAH $\mathcal{A}$ from a data set $D = \{d[1], \dots, d[N]\}$. This learning task is fairly complex, as many aspects are unknown: We are uncertain about the structure of the tree T, the models at the nodes of T, the parameters $\boldsymbol{\theta}$, and the assignment of the instances in D to leaves of T. Hence, the likelihood function has multiple local maxima, and no general method exists for finding the global maximum. In this section, we provide an efficient algorithm for finding a locally optimal $\mathcal{A}$.

To simplify the presentation of our algorithm, we assume that the structure of the CPMs $M_1, \dots, M_m$ is fixed. This reduces the choice of each M_i to a pure numerical optimization problem. The general framework of our algorithm extends to cases where we also have to solve the model selection problem for each M_i, but the computational issues are somewhat different. We omit a discussion for lack of space.

Our algorithm is based on the *Expectation Maximization (EM)* algorithm [2] and the *structural EM* algorithm [4] which extends EM to the problem of model selection. Starting from an initial model, EM iterates the following two steps: The E-step computes the distribution over the unobserved variables given the observed data and the current model. The M-step re-estimates the model parameters by maximizing the expected log likelihood of the data, relative to the distribution computed in the E-step. The SEM algorithm is similar, except that the M-step can also adapt the model structure, and not only its parameters.

[1] Two models are considered identical if $\forall s \in S : P(s \mid M) = P(s \mid M')$.

[2] Care must be taken to ensure that $P(\mathcal{A})$ is a proper probability distribution, but this will always be the case for the choice of ρ we use in this paper. We also note that, if desired, we can modify this prior to incorporate a more traditional parameter prior over the parameters of the M_i's.

In our setting, the unobserved variables (at the global level) consist of the choice of class $C[j] \in \{1, \ldots, k\}$ for each data instance $d[j]$ (as discussed above, $d[j]$ is always generated from a leaf in T). Hence, the E-step computes the probability distribution $P(C[j] = i \mid D, \mathcal{A}) = P(C[j] = i \mid d[j], \mathcal{A})$ for all $1 \leq i \leq N$. In the M-step, we use the completed data, including the class probabilities, to define a new abstraction hierarchy $\mathcal{A}$. This step is quite subtle, and its efficient implementation is at the heart of our algorithm.

When constructing a new hierarchy, our goal is to find the structure of the hierarchy and the parameters of all the M_i's (leaves and internal nodes) that maximize $\log P(A|D)$. This problem, of constructing a tree over a set of points that is not fixed, is very closely related to the *Steiner tree problem* [9], virtually all of whose variants are NP-hard. Somewhat surprisingly, we show that our careful choice of additive prior allows the structure selection problem to be addressed in isolation using global optimization methods. Also surprisingly, we show that, given the structure, the joint M_i parameter optimization problem is convex and therefore we can compute the exact solution very efficiently. These two observations are the key insights behind our algorithm, shown in Fig. 1.

1. **Initialize** $\mathcal{A} = (\{M_1, \ldots, M_k\}, \emptyset)$ and the models at the leaves. Randomly initialize θ.
2. Repeat until convergence:
   (a) T**-step:**
      i. Choose an MST over some subset of $\{v_1, \ldots, v_m\}$, using $\rho(M_i, M_j)$ as edge weights.
      ii. Transform the MST so that $v_1, \ldots, v_k$ become leaves.
   (b) E**-step:** For $j = 1, \ldots, N$, compute the posterior probabilities for the indicator variable $C[j]$. For $1 \leq i \leq k$:
   $$q(C[j] = i) := P(C[j] = i \mid D, \mathcal{A}) \propto P(d[j] \mid M_i)\theta_i.$$
   (c) M**-step:** Update the CPMs and $\boldsymbol{\theta}$. Let $(D, C) = \{d[j], C[j]\}_{j=1}^{N}$. Then:
   $$\begin{aligned} \theta_i &:= \frac{1}{N}\sum_{j=1}^{N} q(C[j] = i) \\ \boldsymbol{M} &:= \arg\max \boldsymbol{E}_q[\log P((D, C), \boldsymbol{M} \mid T)]. \end{aligned}$$

Figure 1: Abstraction Hierarchy Learning Algorithm

The algorithm starts with an empty tree and some initialization for the models at the leaves. The initial model parameters may be selected randomly or, if $k = N$, by assigning each data instance to one of the k leaves. The E-step is the standard one, as described above. Our algorithm partitions the problem of adapting $\mathcal{A}$ into two step. In the *T-step*, we adapt the tree structure relative to the current set of CPMs. In the *M-step*, we adapt the parameters of the CPMs relative to the selected tree structure.

Our goal in the T-step (2a) is to select a good tree structure relative to the current CPMs. Due to the decomposability of $\log P(\mathcal{A})$, the quality of the tree can be measured via the sum of the edge weights $\rho(M_i, M_j)$, for edges (i, j) in the tree; our goal is to find the lowest weight tree. However, our tree structure must keep the same set of leaves $v_1, \ldots, v_k$, but internal nodes can be discarded if that improves the overall score of the hierarchy. It is not difficult to show that this problem is also a variant of the Steiner tree problem. As a heuristic substitute, we follow the lines of [5] and use a minimum spanning tree (MST) algorithm for constructing low-cost trees, where the weight of an edge (i, j) is $\rho(M_i, M_j)$. We can construct an MST over the leaves $v_1, \ldots, v_k$ alone, or over the entire set of nodes $v_1, \ldots, v_m$. Of course, in the resulting tree, some M_i for $1 \leq i \leq k$ may not be leaves. In this case, we perform a transformation that preserves the weight (score) of the tree. Specifically, if M_i is an internal node, we "push" it down the tree by creating a new internal node M' whose CPM is identical to that of M_i, and make M_i a child of M'. As $M' = M_i$, the added cost to the tree is zero. We note that this transformation is not unique, as it depends on the order in which the steps are executed. We therefore consider a pool of trees that include: the previous tree; new MSTs built over $v_1, \ldots, v_k$ using different random resolutions of ambiguities in the weight-preserving transformation; and a set of new MSTs over $v_1, \ldots, v_m$ after deleting internal nodes which end up as leaves of the MST (and are therefore useless), and using different weight-preserving transformations. As we discuss below, our implementation of the M-step is efficient enough that we can carry out the M-step for all the proposed trees in parallel and only keep the tree with the best performance.

In the M-step (2c), we update $\boldsymbol{\theta}$ as well as all the parameters of all of the CPMs $M_1, \ldots, M_m$. It turns out that, under certain conditions, this optimization can be carried out *exactly* for all the parameters of all of the models (both at leaves and at internal nodes). To obtain this condition, both $\rho(M, M')$ and $-\log P(d \mid M_i)$ (for every $d \in S$) must be convex functions in the parameters of M and M'. It is easy to show that ρ is convex for both the quadratic distance function and the symmetric KL-distance mentioned above. The negative log-likelihood is convex for Gaussian and multinomial distributions. In general, however, this assumption is far from trivial, as it also excludes CPMs with hidden variables. (We can still deal with these cases, but not using convex optimization.) A particularly satisfying case occurs when we use a quadratic distance measure ρ and Gaussian CPMs M_i. In this case, the M-step can be carried out entirely in closed form. More generally, under the convexity assumptions, the optimal model parameters can be computed very efficiently. In fact, we can use techniques from optimization theory to provide a very satisfying bound on the number of update steps required to achieve optimality in our objective function up to a certain precision. We omit details for lack of space.

It is easy to show that both the T-step and the M-step increase the expected log-probability $\boldsymbol{E}_{q(C)}[\log P(\mathcal{A}, (D, C))]$. Hence, a simple analysis along the lines of [4] can be used to show that the log-probability $\log P(\mathcal{A} \mid D)$ also increases. As a consequence, we obtain the following theorem:

Theorem 3.1 *The algorithm in Fig. 1 converges to a local maximum of* $\log P(\mathcal{A} \mid D)$.

4 Experimental Results

As discussed earlier, data hierarchies often arise in biological domains. A noteable example is genomic expression data, where the level of mRNA transcript of every gene in the cell can be measured simultaneously using DNA microarray technology. The mRNA transcript of a gene is an intermediate step before producing its final protein product. Thus, genomic expression data provides researchers with much insight towards understanding the overall cellular behavior. The most commonly used method for analyzing this data is clustering, a process which identifies clusters of genes that share similar expression patterns (e.g., [3]). Genes that are similarly expressed are often involved in the same cellular processes. Therefore, clustering suggests functional relationships between clustered genes.

Perhaps the most popular clustering method for genomic expression data to date is hierarchical agglomerative clustering (HAC) [3], which builds a hierarchy among the genes by iteratively merging the closest genes relative to some distance metric. We focus our experimental results on genomic expression data and compare PAH to HAC. We also provide results for a real world text data set.

Synthetic Data. We generated a synthetic data set by sampling from the leaves of a PAH; to make the data realistic, we sampled from a PAH that we learned from a real gene expression data set. To allow a comparison with HAC, we generated one data instance from each leaf. We generated data for 80 (imaginary) genes and 100 experiments, for a total of 8000 measurements. For robustness, we generated 5 different such data sets and ran PAH and HAC for each data set. Since the expression data is continuous, we used Gaussian CPMs M_i and a quadratic distance measure $\rho(M_i, M_j) = \sum_e (M_{ie} - M_{je})^2$, where e ranges over the possible experiments. We used the same distance metric for HAC (note that in HAC the metric is used as the distance between data cases whereas in our algorithm it is used as the distance between models). We note that for HAC, we also tried the Pearson correlation coefficient (more commonly used for gene expression data – see [3]) as a distance metric but its performance was consistently worse.

A good algorithm for learning abstraction hierarchies should recover the true hierarchy as well as possible. To test this, we measured the ability of each method to recover the distances between pairs of instances (genes) in the generating model, where distance here is the length of the path between two genes in the hierarchy. We used the correlation and the L_1 error between the pairwise distances in the original and the learned tree as measures of similiarity. The correlation was 0.72 ± 0.08 for PAH, compared to a much worse 0.27 ± 0.09 for HAC. The average L_1 error was 4.78 ± 1.29 for PAH and 12.46 ± 1.09 for HAC. These results show that PAH recovers an abstraction hierarchy much better than HAC.

Gene Expression Data. We then ran PAH on the data set of [6], who characterized the genomic expression patterns of yeast genes in different experimental conditions. We selected 953 genes with significant changes in expression, using their full set of 93 experiments.

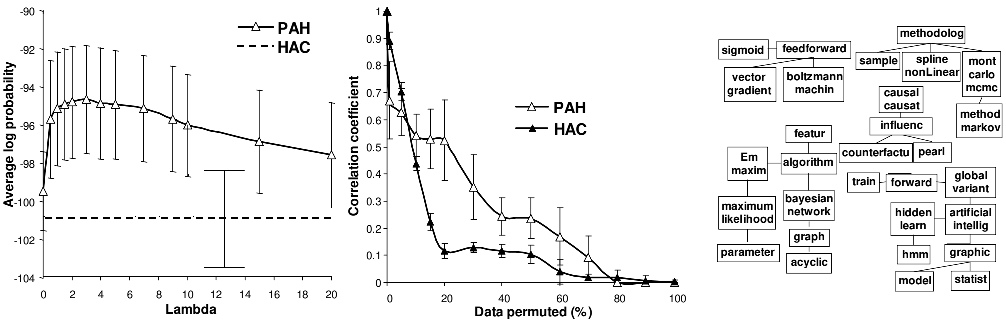
We first measured the ability of the different methods to generalize to unobserved (test) data. This shows how well each method is at capturing the underlying structure in the data. Again, we ran PAH and HAC and evaluated performance using 5 fold cross validation. To test generalization, we need a probabilistic model. To obtain a model from HAC, we created CPMs from the genes that HAC assigned to each internal node. We then learned a model using HAC and PAH for each training set and evaluated its likelihood on the test set. This was done by choosing, for each gene, the best (highest likelihood) CPM among all the nodes in the tree (including internal nodes) and recording the probability $P(g \mid M_{best})$ that this CPM assigns to the gene. For PAH we also used different settings for λ (the coefficient of the penalty term in $P(\mathcal{A})$), which explores the performance in the range of only fitting the data ($\lambda = 0$) and greatly favoring hierarchies in which nearby models are similar (large λ) . The results, summarized in Fig. 2(a), clearly show that PAH generalizes much better to previously unobserved data than HAC and that PAH works best at some tradeoff between fitting the data and generating a hierarchy in which nearby models are similar.

Gene expression measurements are known to be extremely noisy due to imperfections in current DNA microarray technology. Thus, it is crucial that analysis methods for gene expression data be robust to noise as much as possible. Specifically, we want genes that are assigned to nearby nodes in the tree, to be assigned to nearby nodes even in the presence of noise. To test this, we performed the following for each method separately. We learned a model from the original data set and from perturbed data sets in which we permuted a varying percentage of the expression measuments. We then compared the distances (the path length in the tree) between every pair of genes in models learned from the original data and models learned from perturbed data sets. The results are shown in Fig. 2(b), demonstrating that PAH preserves the pairwise distances extremely well even when 20% of the data is perturbed (and performs reasonably well for $30 - 40\%$ permutation), while HAC completely deteriorates when 20% of the data is permuted.

Text Dataset. As a final example, we ran PAH on 350 documents from the Probabilistic Methods category in the Cora dataset (`cora.whizbang.com`) and learned hierarchies among the (stemmed) words. We constructed a vector for each word with an entry for each document whose value is the TFIDF-weighted frequency of the word within the document. Fig. 2(c) shows parts of the learned hierarchy, consisting of 441 nodes, where we list high confidence words for each node. Interestingly, PAH organized related words into the same region of the tree. Within each region, many words were arranged in a way which is consistent with our intuitive notion of abstraction.

5 Discussion

We presented probabilistic abstraction hierarchies, a general framework for learning abstraction hierarchies from data, which relates different classes in the hierarchy by a



(a) (b) (c)

Figure 2: (a) Generalization to test data (b) Robustness to noise (c) Word hierarchy learned

tree whose nodes correspond to class-specific probability models (CPMs). We utilize a Bayesian approach, where the prior favors hierarchies in which nearby classes have similar data distributions, by penalizing the distance between neighboring CPMs. Our approach leads naturally to a form of parameter smoothing, and provides much better generalization for test data and robustness to noise than other clustering approaches.

A unique feature of PAH is the use of global optimization steps for constructing the hierarchy and for finding the optimal setting of the entire set of parameters. This feature differentiates us from many other approaches that build hierarchies by local improvements of the objective function or approaches that optimize a fixed hierarchy [8]. The global optimization steps help in avoiding local maxima and in reducing sensitivity to noise.

In principle, we can use any probabilistic model for the CPM as long as it defines a probability distribution over the state space. We plan to use this generality to apply PAH to more complex problems, such as clustering proteins based on their amino acid sequence using profile HMMs [10].

References

[1] P. Cheeseman and J. Stutz. *Bayesian Classification (AutoClass): Theory and Results*. AAAI Press, 1995.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, B 39:1–39, 1977.

[3] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *PNAS*, 95:14863–68, 1998.

[4] N. Friedman. The Bayesian structural EM algorithm. In *Proc. UAI*, 1998.

[5] N. Friedman, M. Ninio, I. Pe'er, and T. Pupko. A structural EM algorithm for phylogentic inference. In *Proc. RECOMB*, 2001.

[6] A.P. Gasch, P.T. Spellman, C.M. Kao, O.Carmel-Harel, M.B. Eisen, G.Storz, D.Botstein, and P.O. Brown. Genomic expression program in the response of yeast cells to environmental changes. *Mol. Bio. Cell*, 11:4241–4257, 2000.

[7] L. Getoor, D. Koller, and N. Friedman. From instances to classes in probabilistic relational models. In *Proc. ICML Workshop*, 2000.

[8] T. Hofmann. The cluster-abstraction model: Unsupervised learning of topic hierarchies from text data. In *Proc. IJCAI*, 1999.

[9] F.K. Hwang, D.S.Richards, and P. Winter. *The Steiner Tree Problem*. Annals of Discrete Mathematics, Vol. 53, North-Holland, 1992.

[10] A. Krogh, M. Brown, S. Mian, K. Sjolander, and D. Haussler. Hidden markov models in computational biology: Applications to protein modeling. *Mol. Biology*, 235:1501–1531, 1994.

[11] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proc. ICML*, 1998.

[12] M. Meila and M.I. Jordan. Learning with mixtures of trees. *Machine Learning*, 1:1–48, 2000.