Abstraction Augmented Markov Models

Cornelia Caragea
Computer Science
Iowa State University
cornelia@cs.iastate.edu

Adrian Silvescu
Computer Science
Iowa State University
silvescu@cs.iastate.edu

Doina Caragea
Computer and Information Sciences
Kansas State University
dcaragea@ksu.edu

Vasant Honavar
Computer Science
Iowa State University
honavar@cs.iastate.edu

***Abstract*—High accuracy sequence classification often requires the use of higher order Markov models (MMs). However, the number of MM parameters increases exponentially with the range of direct dependencies between sequence elements, thereby increasing the risk of *overfitting* when the data set is limited in size. We present abstraction augmented Markov models (AAMMs) that effectively reduce the number of numeric parameters of k^{th} order MMs by successively grouping strings of length k (i.e., k-grams) into abstraction hierarchies. We evaluate AAMMs on three protein subcellular localization prediction tasks. The results of our experiments show that *abstraction* makes it possible to construct predictive models that use significantly smaller number of features (by one to three orders of magnitude) as compared to MMs. AAMMs are competitive with and, in some cases, significantly outperform MMs. Moreover, the results show that AAMMs often perform significantly better than variable order Markov models, such as decomposed context tree weighting, prediction by partial match, and probabilistic suffix trees.**

***Keywords*-Markov models; abstraction; sequence classification.**

I. Introduction

Many real-world problems, e.g. protein function or protein subcellular localization prediction, can be cast as *sequence classification* problems (1). Markov models (MMs), which capture dependencies between neighboring sequence elements, are among the most widely used generative models of sequence data (2), (3). In a k^{th} order MM, the sequence elements satisfy the *Markov property*: each element is independent of the rest given k preceding elements (called *parents*). MMs have been successfully applied in many applications including natural language processing (3) and molecular sequence classification (2). One of the main disadvantages of MMs in practice is that the number of MM parameters increases exponentially with the range k of direct dependencies, thereby increasing the risk of *overfitting* when the data set is limited in size.

Against this background, we present abstraction augmented Markov models (AAMMs) aimed at addressing these difficulties. AAMM's advantages are as follows:

- AAMMs effectively reduce the number of numeric parameters of MMs through *abstraction*. Specifically, AAMMs learn an abstraction hierarchy over the set of unique k-grams, i.e., substrings of length k, extracted from the training data. An abstraction hierarchy over such a set is a tree such that the leaf nodes correspond to singleton sets containing individual k-grams, and the internal nodes correspond to abstractions or groupings of "similar" k-grams. The procedure for constructing abstraction hierarchies is based on hierarchical agglomerative clustering. At each step, two abstractions are merged together if they result in the least loss in mutual information with respect to the next element in the sequence. An m-cut or level of abstraction through the resulting abstraction hierarchy is a set of m nodes that form a partition of the set of k-grams. An m-cut specifies an AAMM where the m abstractions are used as "features" in the classification model (with m being much smaller than the number of unique k-grams).
- *Abstraction* acts as a *regularizer* that helps minimize *overfitting* (through parameter smoothing) when the training set is limited in size. Hence, AAMMs can yield more robust models as compared to MMs.

We evaluate AAMMs on three protein subcellular localization prediction tasks. The results of our experiments show that AAMMs are able to use significantly smaller number of features (by one to three orders of magnitude) as compared to MMs. AAMMs often yield significantly more accurate classifiers than MMs. Moreover, the results show that AAMMs often perform significantly better than variable order Markov models (VMMs) (4), such as decomposed context tree weighting, prediction by partial match, and probabilistic suffix trees.

The rest of the paper is organized as follows: Section 2 introduces AAMMs. Section 3 presents experimental design and results and Section 4 concludes with a summary and discussion.

II. From Markov Models To Abstraction Augmented Markov Models

Before introducing *abstraction augmented Markov models*, we briefly review Markov models.

A. Markov Models

Let $\mathbf{x} = x_0 \cdots x_{n-1}$ be a sequence over a finite alphabet $\mathcal{X}$, $\mathbf{x} \in \mathcal{X}^*$, and let X_i, for $i = 0, \cdots, n-1$, denote the random variables corresponding to the sequence elements x_i.

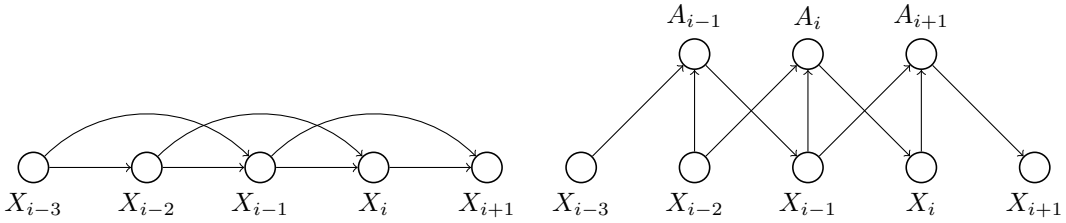


Figure 1: (a) 2^{nd} order Markov model; (b) 2^{nd} order abstraction augmented Markov model

In a k^{th} order Markov model (MM), the sequence elements satisfy the *Markov property*:

$$X_i \perp\!\!\!\perp \{X_0, \cdots, X_{i-k-1}\} \,|\, \{X_{i-k}, \cdots, X_{i-1}\}. \quad (1)$$

That is, X_i is conditionally independent of $X_0, \cdots, X_{i-k-1}$ given $X_{i-k}, \cdots, X_{i-1}$ for $i = k, \cdots, n-1$. $X_{i-k}, \cdots, X_{i-1}$ are called *parents* of X_i. Hence, under a k^{th} order MM, the joint probability of $\mathbf{X} = \{X_0, \cdots, X_{n-1}\}$ can be factorized as follows:

$$p(\mathbf{X}) = p(X_0, \cdots, X_{k-1}) \prod_{i=k}^{n-1} p(X_i | X_{i-k}, \cdots, X_{i-1}). \quad (2)$$

An MM can be represented as a directed graph where the nodes represent the random variables X_i, and the edges represent direct dependencies between neighboring elements of $\mathbf{x}$. Figure 1a shows the directed graph for a 2^{nd} order MM on a subset of nodes of $\mathbf{x}$: $\{X_{i-3}, \cdots, X_{i+1}\}$.

Let S_{i-1} denote the *parents* $X_{i-k} \cdots X_{i-1}$ of X_i in a k^{th} order MM. The values of S_{i-1} represent instantiations of $X_{i-k} \cdots X_{i-1}$, which are substrings of length k (i.e., k-grams) over the alphabet $\mathcal{X}$. Furthermore, let $\mathcal{S}$ denote the set of k-grams over $\mathcal{X}$, s denote a k-gram in $\mathcal{S}$, and σ a symbol in $\mathcal{X}$. The set of parameters θ that define an MM is: $\theta = \{\theta_{\sigma|s} : \sigma \in \mathcal{X}, s \in \mathcal{S}; \theta_s : s \in \mathcal{S}\}$, where $\theta_{\sigma|s} = p(\sigma|s;\theta)$, $\theta_s = p(s|\theta)$.

The cardinality of $\mathcal{S}$ (i.e., $|\mathcal{S}|$) is $|\mathcal{X}|^k$ and is denoted by N. Hence, the number of parameters of a k^{th} order MM is proportional to N, which grows exponentially with the length k of direct dependencies.

B. Abstraction Augmented Markov Models

Abstraction augmented Markov models (AAMMs) effectively reduce the number of numeric parameters of a k^{th} order MM by grouping k-grams into an abstraction hierarchy.

Definition 1 (Abstraction Hierarchy) *An abstraction hierarchy $\mathcal{T}$ over a set of k-grams $\mathcal{S}$ is a rooted tree such that: (1) the root of $\mathcal{T}$ denotes $\mathcal{S}$; (2) the leaves of $\mathcal{T}$ correspond to singleton sets containing individual k-grams in $\mathcal{S}$; (3) the children of each node (say* a*) correspond to a partition of the set of k-grams denoted by* a*. Thus,* a *denotes an abstraction or grouping of "similar" k-grams.*

Note that each internal node (or *abstraction* a) contains the k-grams at the leaves of the subtree rooted at a.

Definition 2 (m-Cut) *An m-cut γ_m through an abstraction hierarchy $\mathcal{T}$ is a subset of m nodes of $\mathcal{T}$ such that for any leaf $s_i \in \mathcal{S}$, either $s_i \in \gamma_m$ or s_i is a descendant of some node in γ_m. The set of abstractions $\mathcal{A}$ at any given m-cut γ_m forms a partition of $\mathcal{S}$.*

Specifically, an m-cut γ_m partitions the set $\mathcal{S}$ of k-grams into m ($m \leq N$) non-overlapping subsets $\mathcal{A} = \{a_1 : \mathcal{S}_1, \cdots, a_m : \mathcal{S}_m\}$, where a_i denotes the i-th abstraction and $\mathcal{S}_i$ denotes the subset of k-grams that are grouped together into the i-th abstraction based on some similarity measure. Note that $\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_m = \mathcal{S}$ and $\forall 1 \leq i, j \leq m,\ \mathcal{S}_i \cap \mathcal{S}_j = \emptyset$.

AAMMs extend the graphical structure of MMs by introducing new variables A_i that represent abstractions over the values of S_{i-1}, for $i = k, \cdots, n-1$. In AAMMs, each node X_i directly depends on A_i as opposed to S_{i-1} (as in MMs). Figure 1b shows the directed graph for a 2^{nd} order AAMM on a subset of nodes: $\{X_{i-3}, \cdots, X_{i+1}\} \cup \{A_{i-1}, \cdots, A_{i+1}\}$. Each variable A_i takes values in the set of abstractions $\mathcal{A} = \{a_1, \cdots, a_m\}$ corresponding to an m-cut, γ_m, which specifies an AAMM. We model the fact that A_i is an abstraction of S_{i-1} by defining $p(A_i = a_i | S_{i-1} = s_{i-1}) = 1$ if $s_{i-1} \in a_i$, and 0 otherwise, where $s_{i-1} \in \mathcal{S}$ and $a_i \in \mathcal{A}$ represent instantiations of S_{i-1} and A_i, respectively.

Under a k^{th} order AAMM, the joint probability of the entire set of variables $\mathbf{X} \cup \mathbf{A}$ can be factorized as follows:

$$p(\mathbf{X}, \mathbf{A}) = p(S_{k-1}) \cdot \prod_{i=k}^{n-1} p(X_i | A_i) \cdot p(A_i | S_{i-1}). \quad (3)$$

The set of parameters θ of an AAMM is: $\theta = \{\theta_{\sigma|a} : \sigma \in \mathcal{X}, a \in \mathcal{A}; \theta_{a|s} : a \in \mathcal{A}, s \in \mathcal{S}; \theta_s : s \in \mathcal{S}\}$, where $\theta_{\sigma|a} = p(\sigma|a;\theta)$, $\theta_{a|s} = p(a|s;\theta)$, and $\theta_s = p(s|\theta)$.

1) Learning AAMMs: In what follows we show how to learn AAMMs from data. This involves: learning an abstraction hierarchy; and learning model parameters using the resulting abstraction hierarchy.

Learning an Abstraction Hierarchy: The procedure for learning an abstraction hierarchy (AH) over the set $\mathcal{S}$ of k-

Algorithm 1 Abstraction Hierarchy Learning

Input: A set of k-grams $\mathcal{S} = \{s_1, \cdots, s_N\}$; a set of sequences $\mathcal{D} = \{\mathbf{x}_l\}_{l=1,\cdots,|\mathcal{D}|}$, $\mathbf{x}_l \in \mathcal{X}^*$
Output: An abstraction hierarchy $\mathcal{T}$ over $\mathcal{S}$
Initialize $\mathcal{A} = \{a_1 : \{s_1\}, \cdots, a_N : \{s_N\}\}$, and $\mathcal{T} = \{a_1 : \{s_1\}, \cdots, a_N : \{s_N\}\}$
for $w = N + 1$ **to** $2N - 1$ **do**
$(u_{min}, v_{min}) = \arg\min_{u,v \in \mathcal{A}} d_{\mathcal{D}}(a_u, a_v)$
$a_w = a_{u_{min}} \cup a_{v_{min}}$
$\mathcal{A} = \mathcal{A} \backslash \{a_{u_{min}}, a_{v_{min}}\} \cup \{a_w\}$
$\mathcal{T} = \mathcal{T} \cup \{a_w\}$ s.t. $Pa(a_{u_{min}}) = Pa(a_{v_{min}}) = a_w$
end for

grams is shown in Algorithm 1. The *input* consists of the set $\mathcal{S}$ of k-grams and a set $\mathcal{D}$ of sequences over the alphabet $\mathcal{X}$, $\mathcal{D} = \{\mathbf{x}_l\}_{l=1,\cdots,|\mathcal{D}|}$. The *output* is an AH $\mathcal{T}$ over $\mathcal{S}$.

The algorithm starts by initializing the set of abstractions $\mathcal{A}$ such that each abstraction $a_i \in \mathcal{A}$ corresponds to a k-gram $s_i \in \mathcal{S}$, $i = 1, \cdots, N$ (the leaves of $\mathcal{T}$ are initialized with elements of $\mathcal{S}$). The algorithm recursively merges pairs of abstractions that are most "similar" to each other and terminates with an AH $\mathcal{T}$ after $N - 1$ steps. We store $\mathcal{T}$ in a last-in-first-out (LIFO) stack. For a given choice of the size m of an m-cut through $\mathcal{T}$, we can extract the set of abstractions that specifies an AAMM, by discarding the top $m - 1$ elements from the stack.

Next we introduce a measure of similarity between a pair of abstractions. We consider two abstractions to be "similar" if *they occur within similar contexts*.

Context of an abstraction. We define the *context of a k-gram $s \in \mathcal{S}$* to be the conditional probability distribution $p(X_i|s)$ of the sequence element X_i that "follows" the k-gram s. The estimate $\hat{p}(X_i|s)$ of $p(X_i|s)$ can be obtained from the data set $\mathcal{D}$ of sequences as follows:

$$\hat{p}(X_i|s) = \left[\frac{1 + \sum_{l=1}^{|\mathcal{D}|} \#[s\sigma, \mathbf{x}_l]}{|\mathcal{X}| + \sum_{\sigma' \in \mathcal{X}} \sum_{l=1}^{|\mathcal{D}|} \#[s\sigma', \mathbf{x}_l]} \right]_{\sigma \in \mathcal{X}} \quad (4)$$

where $\#[s\sigma, \mathbf{x}_l]$ represents the number of times the symbol σ "follows" the k-gram s in the sequence $\mathbf{x}_l$.

The *context of an abstraction* a (i.e., a set of k-grams $a = \{s_1, \cdots, s_{|a|}\}$) is obtained using a weighted aggregation of the contexts of its constituent k-grams. The weights are chosen to ensure that such aggregation yields a *proper* probability distribution. That is,

$$\hat{p}(X_i|a) = \sum_{t=1}^{|a|} \frac{\#s_t}{\sum_{t=1}^{|a|} \#s_t} \cdot \hat{p}(X_i|s_t), \quad (5)$$

where $\#s_t = 1 + \sum_{l=1}^{|\mathcal{D}|} \#[s_t, \mathbf{x}_l]$.

From the preceding definitions it follows that $p(X_i = \sigma|a)$ corresponds to the conditional probability that the symbol σ, $\sigma \in \mathcal{X}$, "follows" *some* k-gram $s_t \in a$.

Distance between abstractions. We proceed to define a *distance* between a pair of abstractions a_u and a_v, denoted by $d_{\mathcal{D}}(a_u, a_v)$. As we shall see below, the definition of d ensures that, at each step, Algorithm 1 selects a pair of abstractions to merge such that the loss of information resulting from the merger is minimized.

The reduction, due to a single step of Algorithm 1, in mutual information between a node X_i and its *parent* A_i in an AAMM (see Figure 1b) can be calculated as follows: Let γ_m be an m-cut through the AH $\mathcal{T}$ and γ_{m-1} be the $(m-1)$-cut through $\mathcal{T}$ that results after the merger of a_u and a_v into a_w, i.e., $\{a_u, a_v\} \rightarrow a_w$. Let $\mathcal{A}_m$ and $\mathcal{A}_{m-1}$ denote the sets of abstractions corresponding to γ_m and γ_{m-1}, respectively. Furthermore, let π_u and π_v denote the prior probabilities of a_u and a_v in the merger a_w, i.e., $\pi_u = \frac{p(a_u)}{p(a_u)+p(a_v)}$ and $\pi_v = \frac{p(a_v)}{p(a_u)+p(a_v)}$ [1].

Proposition 1: *The reduction in the mutual information between each variable X_i and its parent A_i, due to the merger of a_u and a_v into a_w is given by $\delta I(\{a_u, a_v\}, a_w) = (p(a_u) + p(a_v)) \cdot JS_{\pi_u,\pi_v}(p(X_i|a_u), p(X_i|a_v)) \geq 0$, where $JS_{\pi_u,\pi_v}(p(X_i|a_u), p(X_i|a_v))$ represents the weighted Jensen-Shannon divergence (5) between two probability distributions $p(X_i|a_u)$ and $p(X_i|a_v)$ with weights π_u and π_v, respectively.*

We define the distance between two abstractions a_u and a_v in $\mathcal{D}$ as follows:

$$d_{\mathcal{D}}(a_u, a_v) \quad = \quad \delta I(\{a_u, a_v\}, a_w) \text{ where } a_w = \{a_u \cup a_v\}.$$

The effect of one merge of Algorithm 1 on the log likelihood of the data is given by the following proposition.

Proposition 2: *The reduction in the log likelihood of the data $\mathcal{D}$ given an AAMM based on the merger $\{a_u, a_v\} \rightarrow a_w$ is given by $\delta LL(\{a_u, a_v\}, a_w) = M \cdot (p(a_u) + p(a_v)) \cdot JS_{\pi_u,\pi_v}(p(X_i|a_u), p(X_i|a_v)) \geq 0$, where M is the cardinality of the* multiset *of $(k+1)$-grams in $\mathcal{D}$.* (See **Appendix B** for the proof sketch of **Propositions 1** and **2**).

Algorithm Analysis: Recall that $\mathcal{S} = \{s_1, \cdots, s_N\}$ is the set of unique k-grams in $\mathcal{D}$, $N = |\mathcal{S}|$, and that $\mathcal{A} = \{a_1, \cdots, a_m\}$ is the set of constructed abstractions, $m = |\mathcal{A}|$. At each step, the algorithm searches for a pair of abstractions that are most "similar" to each other. The computation of $d_{\mathcal{D}}(a_u, a_v)$ takes $O(|\mathcal{X}|)$ time. Furthermore, at each step, for each $\mathcal{A}_w = \{a_1 : \mathcal{S}_1, \cdots, a_w : \mathcal{S}_w\}$,

[1] The probability $p(a)$ represents the prior probability of an abstraction a. The estimate $\hat{p}(a)$ of $p(a)$ can be obtained from $\mathcal{D}$ as follows:

$$\hat{p}(a) = \frac{1 + \sum_{l=1}^{|\mathcal{D}|} \#[a, \mathbf{x}_l]}{|\mathcal{A}| + \sum_{a' \in \mathcal{A}} \sum_{l=1}^{|\mathcal{D}|} \#[a', \mathbf{x}_l]},$$

where $\#[a, \mathbf{x}_l]$ is the number of times a occurs in $\mathbf{x}_l$ (Note that $a = \{s_1, \cdots, s_{|a|}\}$. If we "abstract out" the difference between all the k-grams $s_1, \cdots, s_{|a|}$ in a and replace each of their occurrences in data $\mathcal{D}$ by a, then $\#a = \sum_{t=1}^{|a|} \#s_t$).

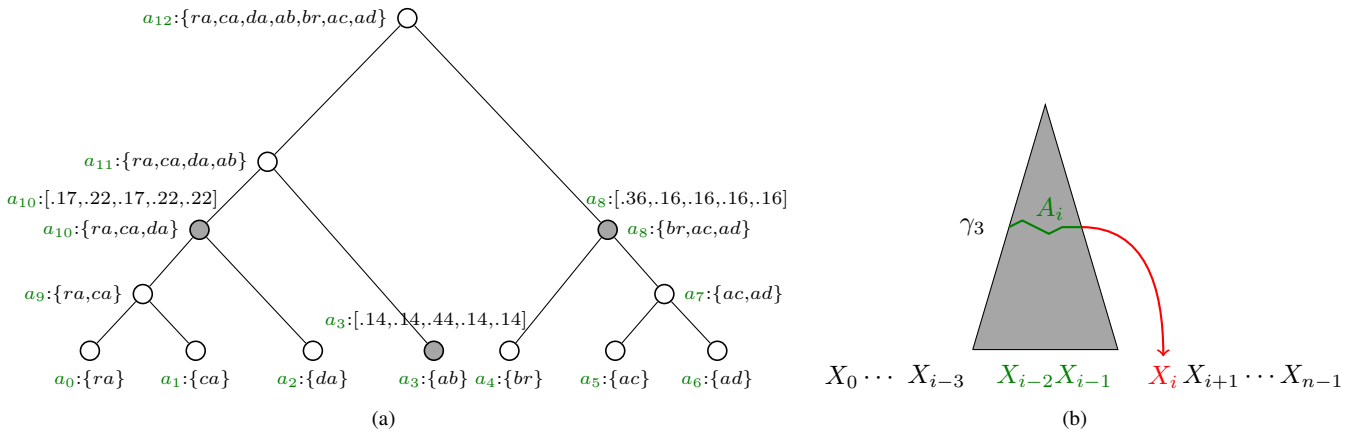


Figure 2: (a) An abstraction hierarchy $\mathcal{T}$ on a set $\mathcal{S} = \{ra, ca, da, ab, br, ac, ad\}$ of 2-grams over the alphabet $\mathcal{X} = \{a, b, c, d, r\}$. $\mathcal{T}$ is learned from the training sequence $abracadabra$. The subset of nodes $\mathcal{A} = \{a_{10}, a_3, a_8\}$ represents a 3-cut γ_3 through $\mathcal{T}$; (b) The computation of $p(\mathbf{x} = x_0, \cdots, x_{n-1})$ given the abstraction hierarchy $\mathcal{T}$ and the cut γ_3.

$w = N, \cdots, m+1$, there are $\frac{w(w-1)}{2}$ possible pairs of abstractions to consider. However, the computational time can be reduced by a factor of N by precomputing the distances $d_{\mathcal{D}}(a_u, a_v)$ between each pair of (trivial) a_u and a_v in $\mathcal{A}_N$, and then, at each step, updating only the distances between pairs containing $a_{u_{min}}$ and $a_{v_{min}}$. Thus, the time complexity of Algorithm 1 is $O(N^2|\mathcal{X}|)$.

Next we show how to learn AAMM parameters from data using the resulting abstraction hierarchy $\mathcal{T}$.

Learning AAMM Parameters: AAMMs are completely observable graphical models (i.e. there are no hidden variables). Given a training set $\mathcal{D} = \{\mathbf{x}_l\}_{l=1,\cdots,|\mathcal{D}|}$, and a set of abstractions $\mathcal{A}$ corresponding to an m-cut, γ_m, through the resulting AH $\mathcal{T}$, learning an AAMM reduces to estimating the set of parameters θ from $\mathcal{D}$, denoted by $\hat{\theta}$, using maximum likelihood estimation (6). This can be done as follows: use Equation (4) to obtain the estimates $\left[\hat{\theta}_{\sigma|s}\right]_{\sigma\in\mathcal{X}}$ of $\left[\theta_{\sigma|s}\right]_{\sigma\in\mathcal{X}}$ for any k-gram $s \in \mathcal{S}$ (note that these estimates correspond to the estimates $\left[\hat{\theta}_{\sigma|a}\right]_{\sigma\in\mathcal{X}}$ when $a = \{s\}$, i.e., the leaf level in the AH $\mathcal{T}$). The estimates $\left[\hat{\theta}_{\sigma|a}\right]_{\sigma\in\mathcal{X}}$ of $\left[\theta_{\sigma|a}\right]_{\sigma\in\mathcal{X}}$, when $a = \{s_1, \cdots, s_{|a|}\}$, are a weighted aggregation of the estimates of a's constituent k-grams, i.e.,

$$\hat{\theta}_{\sigma|a} = \sum_{t=1}^{|a|} \frac{\#s_t}{\sum_{t=1}^{|a|} \#s_t} \cdot \hat{\theta}_{\sigma|s_t}, \tag{6}$$

where $\#s_t$ are defined as before. The estimate $\hat{\theta}_s$ of θ_s is obtained from $\mathcal{D}$ as follows:

$$\hat{\theta}_s = \frac{1 + \sum_{l=1}^{|\mathcal{D}|} \#[s, \mathbf{x}_l]}{|\mathcal{S}| + \sum_{s'\in\mathcal{S}} \sum_{l=1}^{|\mathcal{D}|} \#[s', \mathbf{x}_l]}, \tag{7}$$

where $\#[s, \mathbf{x}_l]$ is the number of times s occurs in $\mathbf{x}_l$. We used Laplace correction to obtain smoothed estimates of probabilities.

Given an AH and a choice of the size m of an m-cut, an array of indices of size N (corresponding to the number of unique k-grams extracted from $\mathcal{D}$[2]) is used to specify the membership of k-grams in the abstractions on the m-cut. Hence, the space complexity for storing this array is N. However, the number of parameters of the corresponding AAMM is $m|\mathcal{X}|$, as opposed to $N|\mathcal{X}|$ in the case of MMs ($m \ll N$).

Figure 2a shows an example of an AH $\mathcal{T}$ learned from a training set, which consists of a single sequence $s = abracadabra$ over the set of 2-grams $\mathcal{S} = \{ra, ca, da, ab, br, ac, ad\}$ extracted from s, where the alphabet $\mathcal{X}$ is $\{a, b, c, d, r\}$. In the figure, the subset of nodes $\{a_{10}, a_3, a_8\}$ represents a 3-cut γ_3 through $\mathcal{T}$. The nodes of γ_3 are annotated with the AAMM parameters learned from the same training set of a single sequence $abracadabra$. Thus, the probabilities that the letters a, b, c, d, and r "follow" the abstraction $a_{10} : \{ra, ca, da\}$, i.e., $\left[\hat{\theta}_{\sigma|a_{10}}\right]_{\sigma\in\mathcal{X}}$, are .17, .22, .17, .22, and .22, respectively. (Note that, in practice, $\mathcal{T}$ is learned from a training set consisting of a large number of sequences).

2) Using AAMMs for Classification: Given a new sequence $\mathbf{x} = x_0, \cdots, x_{n-1}$ and an AAMM (corresponding to an m-cut γ_m), $p(\mathbf{x}|\hat{\theta})$ is obtained as follows: initialize $p(\mathbf{x}|\hat{\theta})$ by $\hat{\theta}_{x_0,\cdots,x_{k-1}}$. For each k-gram $x_{i-k}, \cdots, x_{i-1}$ find the abstraction $a_j \in \gamma_m$ it belongs to and retrieve the parameters associated with a_j. Successively multiply $\hat{\theta}_{x_i|a_j}$ for $i = k, \cdots, n-1$ to obtain $p(\mathbf{x}|\hat{\theta})$.

Figure 2b shows how to compute $p(\mathbf{x})$ given the resulting abstraction hierarchy over the set of 2-grams and the cut

[2]The number of unique k-grams is exponential in k. However, for large values of k, many of the k-grams may not appear in the data. Note that the number of unique k-grams is bounded by the size of $\mathcal{D}$, i.e., the number of (non-unique) k-grams in $\mathcal{D}$.

γ_3. For example, $p(abracadabra|\hat{\theta})$, where $\hat{\theta}$ represents the AAMM corresponding to the cut $\{a_{10}, a_3, a_8\}$ in Figure 2a, is obtained as follows:

$$\begin{aligned} p(abracadabra) &= p(ab)p(r|a_3)p(a|a_8)p(c|a_{10})p(a|a_8) \\ &\quad p(d|a_{10})p(a|a_8)p(b|a_{10})p(r|a_3)p(a|a_8) \\ &= 0.18 \cdot 0.14 \cdot 0.36 \cdot 0.17 \cdot 0.36 \\ &\quad 0.22 \cdot 0.36 \cdot 0.22 \cdot 0.14 \cdot 0.36 \end{aligned}$$

AAMMs can be used for classification by learning a model for each class and selecting the model with the highest posterior probability when classifying new data. Specifically, classification of a sequence $\mathbf{x}$ requires computation of conditional probability $p(c_j|\mathbf{x};\hat{\theta})$, for each class $c_j \in \mathcal{C}$, where $\mathcal{C}$ is the set of possible classes. By applying Bayes rule, we obtain:

$$p(c_j|\mathbf{x};\hat{\theta}) \propto p(\mathbf{x}|c_j;\hat{\theta})p(c_j|\hat{\theta}). \quad (8)$$

The class with the highest posterior probability, $\arg\max_j p(c_j|\mathbf{x};\hat{\theta})$ is assigned to $\mathbf{x}$.

III. Experiments and Results

A. Experimental Design

Our experiments are designed to explore the following questions: (i) How does the performance of AAMMs compare with that of MMs and Naïve Bayes (NB) classifiers, given that AAMMs effectively reduce the number of numeric parameters of MMs through *abstraction*? (ii) What is the effect of the algorithms for learning AHs on the quality of the predictions made by AAMMs? (iii) How does the performance of AAMMs compare with that of variable order Markov models (VMMs) that use more compact representations of the abstraction hierarchies compared to AAMMs?

To answer the first question, we trained AAMMs for values of m that range from 1 to N, where m is the cardinality of the set of abstractions $\mathcal{A}_m$ used as "features" in the classification model, and N is the number of unique k-grams, and compared the performance of AAMM with that of MM and NB over the entire range from 1 to N.

To answer the second question, we compared our AAMM clustering algorithm with agglomerative information bottleneck (AIB) introduced by Slonim and Tishby (7). The primary difference between our AAMM clustering algorithm and AIB is in the criterion used to cluster the k-grams, i.e., in AAMM, the k-grams are clustered based on the similarity between the conditional distributions of X_i given the k-grams, where X_i takes values in $\mathcal{X}$; in AIB, the k-grams are clustered based on the similarity between the conditional distributions of the class variable C given the k-grams, where C takes values in $\mathcal{C}$.

We learned AHs from *training sequences* as follows: (i) a class-specific AH for each class using our AAMM clustering algorithm (from sequences belonging to that class); (ii) a class-independent AH using our AAMM clustering algorithm (from all training sequences, independent of the class variable); and (iii) an AH using the AIB clustering algorithm (from all sequences). In each case, we learned AAMM parameters for each class (from sequences in that class). We compared the performance of AAMMs (using different clustering algorithms) over the entire range from 1 to N.

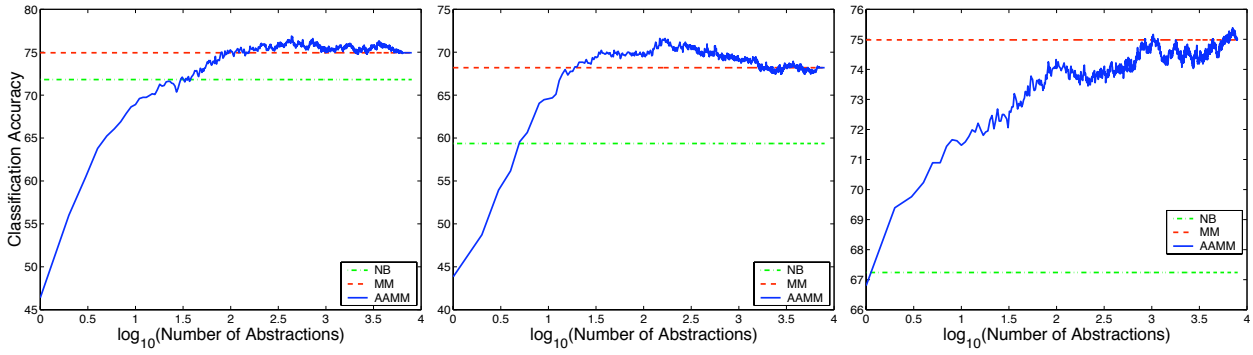
To answer the third question, we trained AAMMs for values of m ranging from 1 to N and compared their performance with that of VMM-type learning algorithms (4), including Lempel-Ziv 78 (LZ78); an improved version of LZ78, namely LZ-MS; decomposed context tree weighting (DE-CTW); prediction by partial match (PPM-C); and probabilistic suffix tree (PST). In our experiments, the AAMM order is $k = 3$. We set the parameters of the VMM-type algorithms as follows: input shifting $S = 2$, back-shift parsing $M = 1$ for LZ-MS; the upper bound k on the Markov order for DE-CTW, PPM-C, and PST is set to 3. In addition, for PST, the other parameters are set as in (4), namely $p_{min} = 0.001$, $\alpha = 0$, $\gamma = 0.0001$, and $r = 1.05$. (see Appendix A for a brief description of these five learning algorithms and an explanation of parameters). For the VMM-type learning algorithms, we have used the VMM implementation of Begleiter et al., 2004 (4).

We present results of experiments on three protein subcellular localization data sets: **psortNeg**[3] introduced in (8), **plant**, and **non-plant**[4] introduced in (9). The **psortNeg** data set is extracted from PSORTdb v.2.0 Gram-negative sequences, which contains experimentally verified localization sites. Our data set consists of all proteins that belong to exactly one of the following five classes: *cytoplasm* (278), *cytoplasmic membrane* (309), *periplasm* (276), *outer membrane* (391) and *extracellular* (190). The total number of examples (proteins) in this data set is 1444. The **plant** data set contains 940 examples belonging to one of the following four classes: *chloroplast* (141), *mitochondrial* (368), *secretory pathway/signal peptide* (269) and *other* (consisting of 54 examples with label nuclear and 108 examples with label cytosolic). The **non-plant** data set contains 2738 examples, each in one of the following three classes: *mitochondrial* (361), *secretory pathway/signal peptide* (715) and *other* (consisting of 1224 examples labeled nuclear and 438 examples labeled cytosolic).

For all of the experiments, we report the average classification accuracy obtained in a 5-fold cross-validation experiment. We define the relative reduction in classification error between two classifiers to be the difference in error divided by the larger of the two error rates. To test the statistical significance of our results, we used the 5-fold cross-validated paired t test for the difference in two classification accuracies

[3] www.psort.org/dataset/datasetv2.html

[4] www.cbs.dtu.dk/services/TargetP/datasets/datasets.php



(a) **psortNeg** (b) **plant** (c) **non-plant**

Figure 3: Comparison of abstraction augmented Markov model (AAMM) with Markov model (MM) and Naïve Bayes (NB) on (a) **psortNeg**, (b) **plant**, and (c) **non-plant** data sets, respectively. The x axis shows the number of abstractions m, used as "features" in the classification model, on a logarithmic scale.

(10). The null hypothesis (i.e., two learning algorithms $\mathcal{M}_1$ and $\mathcal{M}_2$ have the same accuracy on the same test set) can be rejected if $|t(\mathcal{M}_1, \mathcal{M}_2)| > t_{4,0.975} = 2.776$. We abbreviate $|t(\mathcal{M}_1, \mathcal{M}_2)|$ by $|t|$ in what follows.

B. Results

We trained AAMMs and MMs using 3-grams extracted from the data. For **psortNeg**, **plant**, and **non-plant** data sets, the numbers of 3-grams are 7970, 7965, and 7999 respectively.

Comparison of AAMMs with MMs and NB. Figure 3 shows results of the comparison of AAMMs with MMs on all three data sets considered in this study. As can be seen in the figure, AAMM matches the performance of MM with substantially smaller number of abstractions. Specifically, the performance of MM trained using approximately 8000 3-grams is matched by that of AAMM trained using only 79, 19 and 855 abstractions on the **psortNeg**, **plant**, and **non-plant** data sets, respectively. On the **psortNeg** and **non-plant** data sets, AAMM has performance similar to that of MM over a broad range of choices of m. On the **plant** data set, AAMM significantly outperforms MM for many choices of m. For example, with only 168 abstractions, AAMM achieves its highest accuracy of 71.59% as compared to MM which achieves an accuracy of 68.19% with $N = 7965$ ($|t| = 3.03$). This represents 13% reduction in classification error. Not surprisingly, when $m = N$, the performance of AAMMs is the same as that of MMs (AAMM trained using N abstractions and MM are exactly the same models).

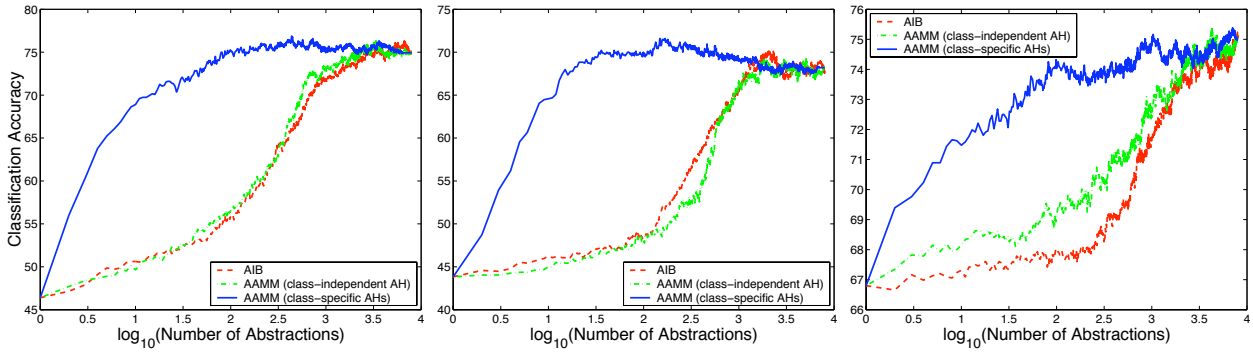
We conclude that AAMMs can match and, in some cases, exceed the performance of MMs using significantly smaller number of abstractions (by one to three orders of magnitude). AAMMs could provide more robust estimates of model parameters than MMs, and hence, help minimize *overfitting*.

Figure 3 also shows the comparison of AAMM with NB trained using a "bag of letters" feature representation. As can be seen, except for a few values of m ($m < 18$, $m < 5$, and $m < 2$ on **psortNeg**, **plant**, and **non-plant**, respectively), AAMM significantly outperforms NB (for any other choices of m). MM is superior in performance to NB on all data sets.

Comparison of AAMM clustering algorithm with Agglomerative Information Bottleneck. Figure 4 shows, on all three data sets, results of the comparison of AAMMs trained based on (i) class-specific AHs, with one AH for each class, (ii) a single class-independent AH, and (iii) an AH produced using AIB (7). As can be seen in the figure, AAMMs trained based on class-specific AHs generated by the clustering algorithm proposed here significantly outperform AAMMs trained based on an AH generated by AIB, over a broad range of values of m (from 1 to 1000). For example, on the **plant** data set, with $m = 100$, the accuracy of AAMM based on our clustering algorithm is 69.57%, whereas that of AIB-clustering based AAMM is 48.29% ($|t| = 11.31$). This represents 42% reduction in classification error. As expected, AAMMs trained using class-specific AHs significantly outperform AAMMs trained using a class-independent AH on all three data sets.

We conclude that organizing k-grams in an AH based on the conditional distribution of the next element in the sequence rather than the conditional distribution of the class given the k-grams produces AHs that are better suited for AAMMs, and hence, result in better performing AAMMs.

Comparison of AAMMs with VMM-type learning algorithms. Table I summarizes, on all three data sets, the results of the comparison of AAMMs with five VMM-type learning algorithms: Lempel-Ziv 78 (LZ78); an improved version of LZ78, namely LZ-MS; decomposed context tree weighting (DE-CTW); prediction by partial match (PPM-C); and probabilistic suffix tree (PST). For AAMMs, we show



(a) **psortNeg** (b) **plant** (c) **non-plant**

Figure 4: Comparison of the AAMM clustering algorithm with the Agglomerative Information Bottleneck on (a) **psortNeg**, (b) **plant**, and (c) **non-plant** data sets, respectively. The x axis shows the number of abstractions m on a logarithmic scale.

Data sets	**LZ78**	**LZ-MS**	**DE-CTW**	**PPM-C**	**PST**	**AAMM**
psortNeg	0.67±0.012	0.69±0.014	0.74±0.008	0.75±0.006	0.76±0.006	**0.77±0.007**
plant	0.62±0.017	0.68±0.019	0.55±0.032	**0.72±0.019**	0.66±0.016	**0.72±0.015**
non-plant	0.67±0.006	0.70±0.005	0.68±0.018	0.73±0.009	**0.79±0.007**	0.75±0.006

Table I: Classification accuracy ± SEM of AAMMs and VMM-type learning algorithms on **psortNeg**, **plant**, and **non-plant** data sets (SEM = standard error of the means).

the best classification accuracy over the entire range of values of m, on each data set. The values of m where AAMM reaches the best classification accuracy are: 438, 168, 7070 on **psortNeg**, **plant**, **non-plant** data sets, respectively.

As can be seen in the table, AAMM significantly outperforms LZ78, LZ-MS, and DE-CTW on all three data sets ($p < 0.05$). AAMM significantly outperforms PPM-C on **psortNeg** ($|t| = 4.973$), and **non-plant** ($|t| = 3.099$), and has the same performance as PPM-C on **plant**. Furthermore, AAMM significantly outperforms PST on **plant** ($|t| = 4.163$), and is comparable in performance with PST on **psortNeg** (the null hypothesis is not rejected). On **non-plant**, PST significantly outperforms AAMM ($|t| = 4.433$).

We conclude that AAMMs are competitive with, and often significantly outperform, VMM-type learning algorithms on the protein subcellular localization prediction task.

IV. Summary and Discussion

A. Summary

We have presented abstraction augmented Markov models that simplify the data representation used by the standard Markov models. The results of our experiments on three protein subcellular localization data sets (**psortNeg**, **plant**, and **non-plant**) have shown that:

- Organizing the set of k-grams in a hierarchy using *abstraction* makes it possible to construct predictive models that use significantly smaller number of features (by one to three orders of magnitude) as compared to MMs (which is exponential in k);
- While *abstraction* helps reduce the number of MM parameters, the performance of AAMMs is similar, and in some cases, significantly better than that of MMs;
- AAMMs are competitive with, and in many cases, significantly outperform variable order Markov models.

These conclusions are supported by results of additional experiments on three other data sets compiled from the *Structural Classification of Proteins* (SCOP) database: **E**, **F**, and **G SCOP classes** that have been used by other authors (4) (data not shown due to space limitation).

B. Related Work

Several authors have used abstraction hierarchies over *classes* to improve the performance of classifiers (e.g., (11; 12). Others have explored the use of abstraction hierarchies to learn compact predictive models (13; 14). Slonim and Tishby (7), Baker and McCallum (15), and Silvescu et al. (16) have generated abstraction hierarchies over features or k-grams based on the similarity of the probability distributions of the classes conditioned on the features or k-grams (respectively) and used the resulting abstract features to train classifiers (15; 16). In contrast, the focus of this paper is on abstraction hierarchies that group k-grams (or more generally their abstractions) based on the similarity of the probability distributions of each letter of the alphabet conditioned on the abstractions, and the use of the resulting abstraction hierarchies over k-grams to construct generative models from sequence data.

Begleiter et al. (4) (and papers cited therein) have examined and compared several methods for prediction using

variable order MMs (VMMs), including probabilistic suffix trees (PSTs) (17). PSTs can be viewed as a variant of AAMMs wherein the abstractions are constrained to share suffixes. Hence, the clustering of k-grams in PSTs can be represented more compactly compared to that of AAMMs, which require storing an array of indices of size N to specify the membership of k-grams in the abstractions (or clusters). The results of our experiments show that AAMMs are competitive with VMM-type learning algorithms. Interpolated MMs (18), which recursively combine several fixed-order MMs, capture important sequence patterns that would otherwise be ignored by a single fixed-order MM.

C. Discussion

Abstraction helps reduce the model input size and, at the same time, could potentially improve the statistical estimates of complex models by reducing the number of parameters that need to be estimated from data (hence reducing the risk of *overfitting*). However, one limitation of our abstraction-based approach is that, while it provides simpler models, the simplicity is achieved at the risk of some information loss due to *abstraction*. To trade off the complexity of the model against its predictive accuracy, it would be useful to augment the algorithm so that it can choose an optimal cut in an AH. This can be achieved by designing a scoring function (based on a conditional MDL score), similar to Zhang et al. (13) in the case of Naïve Bayes, to guide a top-down search for an optimal cut.

It is worth noting that the AHs can be learned using any top-down or bottom-up clustering procedure. However, in this study, we have used a bottom-up approach because it is simple, fast, and allows iterating through all cardinalities, from 1 to N.

Connection between AAMMs and HMMs. An AAMM can be simulated by an appropriately constructed HMM where the number of hidden states is equal to the number of abstractions in the AAMM. However, as a state corresponds to an abstraction over the observable variables, the state is not really "hidden". It can be derived in a feed-forward manner, thus not requiring a Backward Reasoning/Expectation step. This allows a "one pass through the data" learning procedure based on MAP learning (19) once the set of abstractions is chosen. Unlike the learning procedure of HMMs (which involves the Expectation Maximization (EM) algorithm), the AAMM learning procedure is not prone to local maxima, has lower variance as no uncertainty is inherited from the inference performed in the E step, and requires less time.

The overall time complexity of AAMM is $O(N^2 \cdot |\mathcal{X}| + N \cdot |\mathcal{D}|)$, where N is the number of unique k-grams, $|\mathcal{X}|$ is the alphabet size, and $|\mathcal{D}|$ is the data set size, i.e., the number of (non-unique) k-grams in $\mathcal{D}$. The time complexity of HMM EM learning procedure for a fixed number of hidden states is $O(T \cdot |\mathcal{D}| \cdot (|H|^2 + |H| \cdot |\mathcal{X}|))$, where $|H|$ is the number of hidden states and T is the number of EM iterations. Running this procedure for all numbers of hidden states (from 1 to N) requires an overall time of $O(N^2(N + |\mathcal{X}|) \cdot T \cdot |\mathcal{D}|)$. Because $N \ll |\mathcal{D}|$, our algorithm requires at least a factor of $N \cdot T$ less time than HMM. While algorithms that attempt to automatically determine the number of hidden states in HMMs can be used (e.g., based on the Chinese Restaurant Process (20)), they incur additional costs relative to standard HMM, and are prone to the difficulties encountered by hidden variable models. Hence, although less expressive than HMMs, AAMMs are easier to learn.

D. Future directions

Some directions for further research include: (i) design of AA Interpolated MMs (AAIMMs) that extend Interpolated MMs in the same way AAMMs extend MMs; (ii) applications of AAMMs to settings where data have a much richer structure (e.g., images and text); (iii) exploration of alternative clustering algorithms for generating abstraction hierarchies for use with AAMMs; (iv) incorporation of MDL-like criteria for finding an optimal level of abstraction.

ACKNOWLEDGMENT

This research was funded in part by an NSF grant IIS 0711356 to Vasant Honavar and Doina Caragea.

APPENDIX A

In what follows, we briefly describe five algorithms for learning variable order Markov models (VMMs) from a set of training sequences $\mathcal{D}$ over an alphabet $\mathcal{X}$. See (4) for more details and citations therein.

Lempel-Ziv 78 (LZ78): The LZ78 learning algorithm (21), (22) extracts a set $\hat{S}$ of non-overlapping adjacent *phrases* from sequences in $\mathcal{D}$ as follows: $\hat{S}$ initially contains only the empty phrase ϵ; at each step, a new phrase, which extends an existing phrase from $\hat{S}$ by a symbol in $\mathcal{X}$, is added to $\hat{S}$.

The algorithm then constructs a phrase tree $\mathcal{T}$ over $\mathcal{X}$ such that the degree of each internal node is exactly $|\mathcal{X}|$. Initially, $\mathcal{T}$ contains the root and $|\mathcal{X}|$ leaves (i.e., one leaf for each symbol in $\mathcal{X}$). For any phrase $s \in \hat{S}$, start at the root and traverse $\mathcal{T}$ according to s. When a leaf is reached, it is made an internal node by expanding it into $|\mathcal{X}|$ leaf nodes. Each node stores a counter such that the counter of a leaf node is one, and that of an internal node is the sum of the counters stored at the child nodes.

Improved Lempel-Ziv 78 Algorithm (LZ-MS): Two major disadvantages of the LZ78 learning algorithm are: (i) *unreliable estimation of model parameters* when subsequences s of a sequence $\mathbf{x}$ are not parsed, and hence, they are not part of $\mathcal{T}$; (ii) *unreliable computation of* $\hat{p}(\sigma|s)$ when the algorithm ends in a leaf node while traversing $\mathcal{T}$ along the path corresponding to s starting from the root. LZ-MS learning algorithm (23) aims at addressing these disadvantages by introducing two parameters: *input shifting*, denoted by S,

and *back-shift parsing*, denoted by M. The S parameter ensures that more phrases are extracted during learning, whereas the M parameter ensures the existence of a suffix of s when computing $\hat{p}(\sigma|s)$.

Decomposed Context Tree Weighting (DE-CTW): The CTW learning algorithm (24) combines exponentially many VMMs of k-bounded order in an efficient way. The CTW algorithm over *binary* alphabets $\mathcal{X}$ constructs a *perfect binary tree* $\mathcal{T}$ of height k from sequences in $\mathcal{D}$. Each node is labeled with the string s corresponding to the path from this node to the root, and stores two counters, which represent the number of times each symbol in $\mathcal{X}$ (0 or 1) occurs after s in $\mathcal{D}$.

CTW algorithm can be extended to work with multi-alphabets. One approach, called decomposed CTW (DE-CTW), uses a tree-based hierarchical decomposition of the multi-valued prediction problem into binary problems.

Prediction by Partial Match (PPM-C): The PPM learning algorithm (25) requires an upper bound k on the Markov order of the VMM it learns. It constructs a tree $\mathcal{T}$ of maximal depth $k+1$ from sequences in $\mathcal{D}$ as follows: it starts with the root node which corresponds to the empty string ϵ and parses sequences in $\mathcal{D}$, one element at a time; each element x_i in a sequence $\mathbf{x}$ and its k preceding elements $x_{i-k}, \cdots, x_{i-1}$ form a path of length $k+1$ in $\mathcal{T}$. Each node in $\mathcal{T}$ is labeled by a symbol σ and stores a counter. The counter of a node σ on a path $s\sigma$ from the root represents the frequency counts of $s\sigma$ in $\mathcal{D}$, denoted by $\#s\sigma$.

To obtain smoothed estimates of probabilities for any string s, $|s| \leq k$, PPM introduces a new variable, called *escape*, for all symbols in the alphabet that do not appear after s in $\mathcal{D}$ and allocates a probability mass for all these symbols, i.e., $p(escape|s)$. $1 - p(escape|s)$ is distributed among all other symbols that occur after s in $\mathcal{D}$. A successful PPM variant, namely PPM-C, performs mass probability allocation for *escape* and mass probability distribution over the other symbols as follows: $\hat{p}(\sigma|s) = \frac{\#s\sigma}{|\mathcal{X}_s|+\sum_{\sigma' \in \mathcal{X}_s} \#s\sigma'}$, if $\sigma \in \mathcal{X}_s$, $\hat{p}(escape|s) = \frac{|\mathcal{X}_s|}{|\mathcal{X}_s|+\sum_{\sigma' \in \mathcal{X}_s} \#s\sigma'}$, where $|\mathcal{X}_s|$ denotes the set of symbols in $\mathcal{X}$ that occur after s in $\mathcal{D}$.

Probabilistic Suffix Tree (PST): The PST learning algorithm (26) constructs a non empty tree $\mathcal{T}$ over an alphabet $\mathcal{X}$ such that the degree of each node varies between 0 (for the leaves) and $|\mathcal{X}|$ (for the internal nodes). Each edge e is labeled by a single symbol in $\mathcal{X}$, and each node v is labeled by a sub-sequence s that is obtained by the concatenation of edge labels on the path from v up to the root of $\mathcal{T}$.

In the first stage, the PST learning algorithm identifies a set $\hat{S}$ of *candidate suffixes* of length $\leq k$ from $\mathcal{D}$ (k is the maximal length of a suffix), such that the empirical probability of each suffix $s \in \hat{S}$, $\hat{p}(s)$, is above some threshold p_{min}. In the second stage, a candidate suffix s and all its suffixes are added to $\mathcal{T}$ if s satisfies two conditions: s is "meaningful" for some symbol σ (i.e., $\hat{p}(\sigma|s)$ is above some user threshold $(1+\alpha)\gamma_{min}$), and s provides additional information relative to its parent s', where s' is the string obtained from s by deleting the leftmost letter (i.e., $\frac{\hat{p}(\sigma|s)}{\hat{p}(\sigma|s')}$ is greater than a user threshold r or smaller than $1/r$). In the last stage, the probability distribution associated with each node, $p(\sigma|s)$ over $\mathcal{X}$ for each s, are smoothed.

APPENDIX B

Lemma 1: Let X and Z be two random variables such that Z can take on k possible values. Let $p(z_i)$ be the prior distribution of z_i and $p(x|z_i)$ be the conditional distribution of X given z_i for $i = 1, \cdots, k$. Then:

$$JS_{p(z_1),\cdots,p(z_k)}\left(p(x|z_1), \cdots, p(x|z_k)\right)$$
$$= H\left(\sum_{i=1}^{k} p(z_i)p(x|z_i)\right) - \sum_{i=1}^{k} p(z_i)H\left(p(x|z_i)\right) = I(X;Z)$$

where $H(\cdot)$ is Shannon's entropy (5).

Proof of Proposition 1: Without loss of generality, let us assume that the merger is $\{a_1, a_2\} \rightarrow a$. Let $\delta I(\{a_1, a_2\}, a) = I(A^{(\mathcal{A}_m)}, X_i) - I(A^{(\mathcal{A}_{m-1})}, X_i)$ denote the reduction in the mutual information $I(A; X_i)$, where $A^{(\mathcal{A}_m)}$ represents the variable A that takes values in the set $\mathcal{A}_m = \{a_1, \cdots, a_m\}$. We use the above lemma. Hence, $\delta I(\{a_1, a_2\}, a)$

$$\begin{aligned}
&= JS_{p(a_1),p(a_2),\cdots,p(a_m)}\left[p(x_i|a_1), p(x_i|a_2), \cdots, p(x_i|a_m)\right] \\
&\quad - JS_{p(a),\cdots,p(a_m)}\left[p(x_i|a), \cdots, p(x_i|a_m)\right] \\
&= H\left(\sum_{j=1}^{m} p(a_j)p(x_i|a_j)\right) - \sum_{j=1}^{m} p(a_j)H\left(p(x_i|a_j)\right) \\
&\quad - H\left(p(a)p(x_i|a) + \sum_{j=3}^{m} p(a_j)p(x_i|a_j)\right) \\
&\quad + p(a)H\left(p(x_i|a)\right) + \sum_{j=3}^{m} p(a_j)H\left(p(x_i|a_j)\right) \\
&= p(a)H\left(p(x_i|a)\right) - \sum_{j=1}^{2} p(a_j)H\left(p(x_i|a_j)\right) \\
&= p(a)H\left(\frac{1}{p(a)}\sum_{j=1}^{2} p(x_i, a_j)\right) - \sum_{i=1}^{2} p(a_j)H\left(p(x_i|a_j)\right) \\
&= p(a)\left(H\left(\sum_{j=1}^{2} \frac{p(a_j)}{p(a)} p(x_i|a_j)\right) - \sum_{j=1}^{2} \frac{p(a_j)}{p(a)} H\left(p(x_i|a_j)\right)\right) \\
&= \left(p(a_1) + p(a_2)\right) \cdot JS_{\pi_1,\pi_2}(p(X_i|a_1), p(X_i|a_2)).
\end{aligned}$$

Proof of Proposition 2: As in **Proposition 1**, let us assume, without loss of generality, that the merge is $\{a_1, a_2\} \rightarrow a$. Hence, $\#a = \#a_1 + \#a_2$. Furthermore, let $\pi_1 = \frac{p(a_1)}{p(a_1)+p(a_2)}$ and $\pi_2 = \frac{p(a_2)}{p(a_1)+p(a_2)}$.

$$
\begin{aligned}
&\delta LL(\{a_1,a_2\},a) = LL(A^{(\mathcal{A}_m)},X_i) - LL(A^{(\mathcal{A}_{m-1})},X_i)\\
=\;& \sum_{x_i\in\mathcal{X},a_j\in\mathcal{A}_m} \log p(x_i|a_j)^{\#[a_j,x_i]}\\
&- \sum_{x_i\in\mathcal{X},a_j\in\mathcal{A}_{m-1}} \log p(x_i|a_j)^{\#[a_j,x_i]}\\
=\;& \sum_{x_i\in\mathcal{X}} \log p(x_i|a_1)^{\#[a_1,x_i]}\\
&+ \sum_{x_i\in\mathcal{X}} \log p(x_i|a_2)^{\#[a_2,x_i]} - \sum_{x_i\in\mathcal{X}} \log p(x_i|a)^{\#[a,x_i]}\\
=\;& \sum_{x_i\in\mathcal{X}} \#[a_1,x_i]\log p(x_i|a_1) + \sum_{x_i\in\mathcal{X}} \#[a_2,x_i]\log p(x_i|a_2)\\
&- \sum_{x_i\in\mathcal{X}} (\#[a_1,x_i]+\#[a_2,x_i])\cdot \log p(x_i|a_1\cup a_2)\\
=\;& -Mp(a)\sum_{x_i\in\mathcal{X}}\left(\sum_{j=1}^{2}\pi_j p(x_i|a_j)\right)\log\left(\sum_{j=1}^{2}\pi_j p(x_i|a_j)\right)\\
&+M\cdot p(a)\left(\sum_{x_i\in\mathcal{X}}\sum_{j=1}^{2}\pi_j p(x_i|a_j)\log p(x_i|a_j)\right)\\
=\;& Mp(a)\left(H\left(\sum_{j=1}^{2}\pi_j p(x_i|a_j)\right) - \sum_{j=1}^{2}\pi_j H\left(p(x_i|a_j)\right)\right)\\
=\;& M\cdot((p(a_1)+p(a_2))\cdot JS_{\pi_1,\pi_2}(p(X_i|a_1),p(X_i|a_2))
\end{aligned}
$$

where M is the cardinality of the *multiset* of $(k+1)$-grams. We have used that: $p(x_i|a_1\cup a_2) = \pi_1 p(x_i|a_1) + \pi_2 p(x_i|a_2)$, when $a_1\cap a_2 = \phi$. and $\#[a_j,x_i] = p(a_j,x_i)\cdot M = p(x_i|a_j)\cdot p(a_j)\cdot M = p(x_i|a_j)\cdot\pi_j\cdot p(a)\cdot M$.

REFERENCES

[1] P. Baldi and S. Brunak, *Bioinformatics: the Machine Learning Approach*. MIT Press, 2001.

[2] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge Univ., 2004.

[3] E. Charniak, *Statistical Language Learning, Cambridge: 1993*. MIT Press, 1993.

[4] R. Begleiter, R. El-Yaniv, and G. Yona, "On prediction using variable order markov models," *Journal of Artificial Intelligence Res.*, vol. 22, pp. 385–421, 2004.

[5] J. Lin, "Divergence measures based on the Shannon entropy," *IEEE Trans. on Inf. Thr.*, vol. 37, pp. 145–151, 1991.

[6] G. Casella and R. L. Berger, *Statistical Inference.* Duxbury, 2002.

[7] N. Slonim and N. Tishby, "Agglomerative information bottleneck," in *Proc. of NIPS-12*, 1999, pp. 617–623.

[8] J. Gardy, "Psort-b: improving protein subcellular localization prediction for gram-negative bacteria," *Nucleic Ac. Res.*, vol. 31, no. 13, pp. 3613–17, 2003.

[9] O. Emanuelsson, H. Nielsen, S. Brunak, and G. von Heijne, "Predicting subcellular localization of proteins based on their n-terminal amino acid sequence." *J. Mol. Biol.*, vol. 300, pp. 1005–1016, 2000.

[10] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural Computation*, vol. 10, pp. 1895–1923, 1998.

[11] E. Segal, D. Koller, and D. Ormoneit, "Probabilistic abstraction hierarchies," in *Proc. of NIPS*, Vancouver, Canada, 2002, pp. 913–920.

[12] A. K. McCallum, R. Rosenfeld, T. M. Mitchell, and A. Y. Ng, "Improving text classification by shrinkage in a hierarchy of classes," in *Proceedings of ICML*, Madison, US, 1998, pp. 359–367.

[13] J. Zhang, D.-K. Kang, A. Silvescu, and V. Honavar, "Learning accurate and concise naive bayes classifiers from attribute value taxonomies and data," *Knowledge and Information Systems*, vol. 9, pp. 157–179, 2006.

[14] M. desJardins, L. Getoor, and D. Koller, "Using feature hierarchies in bayesian network learning." in *Proc. of SARA*, Springer-Verlag, 2000, pp. 260–270.

[15] L. D. Baker and A. K. McCallum, "Distributional clustering of words for text classification," in *Proc. of ACM SIGIR*. ACM Press, 1998, pp. 96–103.

[16] A. Silvescu, C. Caragea, and V. Honavar, "Combining super-structuring and abstraction on sequence classification," in *ICDM*, 2009, pp. 986–991.

[17] G. Bejerano and G. Yona, "Modeling protein families using probabilistic suffix trees," in *RECOMB '99*. New York, NY, USA: ACM, 1999, pp. 15–24.

[18] F. Jelinek and R. L. Mercer, "Interpolated estimation of markov source parameters from sparse data." *Pattern Recognition in Practice*, pp. 381–397, 1980.

[19] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[20] D. M. Blei, T. Griffiths, M. I. Jordan, and J. Tenenbaum, "Hierarchical topic models and the nested chinese restaurant process." in *Proc. of NIPS*, 2004.

[21] G. G. Langdon, "A note on the zivlempel model for compressing individual sequences," *IEEE Transactions on Information Theory*, vol. 29, pp. 284–287, 1983.

[22] J. Rissanen, "A universal data compression system," *IEEE Trans. on Inf. Thr.*, vol. 29, pp. 656–664, 1983.

[23] M. Nisenson, I. Yariv, R. El-Yaniv, and R. Meir, "Towards behaviometric security systems: Learning to identify a typist," in *ECML/PKDD*, 2003, pp. 363–374.

[24] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens, "The context tree weighting method: Basic properties," *IEEE Trans. on Inf. Thr.*, vol. 41, pp. 653–664, 1995.

[25] J. G. Cleary and I. H. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. on Communications*, vol. 32, pp. 396–402, 1984.

[26] D. Ron, Y. Singer, and N. Tishby, "The power of amnesia: Learning probabilistic automata with variable memory length," in *Machine Learning*, 1996, pp. 117–149.