Clustering Markov States into Equivalence Classes using SVD and Heuristic Search Algorithms

Xianping Ge
Information & Computer Science
University of California, Irvine
Irvine, CA 92697-3430

Sridevi Parise
Information & Computer Science
University of California, Irvine
Irvine, CA 92697-3430

Padhraic Smyth
Information & Computer Science
University of California, Irvine
Irvine, CA 92697-3425

Abstract

This paper investigates the problem of finding a K-state first-order Markov chain that approximates an M-state first-order Markov chain, where K is typically much smaller than M. A variety of greedy heuristic search algorithms that maximize the data likelihood are investigated and found to work well empirically. The proposed algorithms are demonstrated on two applications: learning user models from traces of Unix commands, and word segmentation in language modeling.

1 Introduction

There are numerous applications where we may to want to estimate a Markov chain with a very large number of states M. For example, in modeling of Web navigation patterns M might be the number of Web pages on a Web site; in modeling of text data M could be the number of different words that can occur. In these applications the number of states M may be as large as 10^5 or more. These large values of M will clearly render impractical the direct estimation of $O(M^2)$ transition probability parameters in a first-order Markov model for the data.

In this paper we investigate the general problem of reducing an M-state first-order Markov chain to an approximating K-state Markov chain, where K is much smaller than M. In the reduced chain, each of the original M states are assigned to one of the K state clusters, and K^2 transition probabilities are estimated at the cluster level. The general intuition behind such a reduced-state model is that subsets of the original M states can be viewed as belonging to an equivalence class in terms of their first-order Markov transition behavior. For example, the probability of a particular word in the set A coming next in the text, given that the current word is in some set B, might be the same for all pairs of words in sets A and B. The focus of this paper is to propose and evaluate a number of different learning algorithms that can discover such equivalence classes from observed data.

We begin by showing that this learning problem can be related to singular value decomposition (SVD) of a particular form of block-structured matrix. The SVD method produces an exact solution under certain ideal conditions, but requires very large amounts of data to work effectively in practice. This motivates the proposal of a class of heuristic search algorithms that directly seek K-state models that maximize the likelihood of the observed data. These algorithms view the problem as an assignment problem (iteratively assigning each of the M states to K clusters) and appear to work well in practice even for relatively large values of M and K.

We illustrate the application of these algorithms using two different problems with large alphabet sizes M: (a) learning user models from a set of individuals based on 6 months of Unix command line data, and (b) automatically constructing a K-state Markov chain at the word level to segment English text with word boundaries removed. The resulting reduced Markov chains are shown to clearly capture informative structure in the data.

2 Notation and Model

To avoid confusion about the states in the original M-state Markov model and the states in the new K-state model, we will refer to states in the M-state model as *symbols*. Let $s(y)$ be the state that symbol y is assigned to.

$$P(\text{symbol } y|\text{state } s) = \begin{cases} P_{y|s}, & \text{if } y \in s, \\ 0, & \text{otherwise.} \end{cases}$$

To simplify notation we assume that y_0=START,

y_{T+1}=STOP for any symbol sequence $\mathbf{y} = y_1 \dots y_T$. These two imaginary symbols correspond to the imaginary states START and STOP, respectively.

The set of parameters for the model is

$$\boldsymbol{\theta} = \{\mathbf{A}, \{P_{y|s(y)}\}\},$$

where $\mathbf{A}$ is the transition matrix. Note that the initial state distribution $\boldsymbol{\pi}$ is incorporated into $\mathbf{A}$ as the transition probabilities from the imaginary state START.

Our task is to estimate $\boldsymbol{\theta}$ from a set of symbol sequences $\{\mathbf{y}\}$. The log likelihood is

$$\begin{aligned} \log P(\{\mathbf{y}\}) &= \sum_{\mathbf{y}} \log P(\mathbf{y}) \\ &= \sum_{\mathbf{y}} \sum_{t=1}^{T(\mathbf{y})+1} \Big[\log P\big(s(y_t)|s(y_{t-1})\big) + \log P\big(y_t|s(y_t)\big)\Big] \\ &= \Bigg[\sum_{y,y'} n_{y,y'} \log A_{s(y),s(y')}\Bigg] + \Bigg[\sum_{y} n_y \log P_{y|s(y)}\Bigg] \quad (1) \end{aligned}$$

where

- $T(\mathbf{y})$ is the length of $\mathbf{y}$,
- $n_{y,y'}$ is transition count of the symbol pair (y, y'), i.e., the number of times that symbol y is followed immediately by symbol y',
- $n_y = \sum_{y'} n_{y,y'}$ is count of the symbol y, i.e., the number of occurrences of symbol y.

Using n_y, $n_{y,y'}$, we can define the following counts:

- $n_s = \sum_{y \in s} n_y$ is the count of state s.
- $n_{s,s'} = \sum_{\substack{y \in s \\ y' \in s'}} n_{y,y'}$ is the transition count from state s to state s'.
- $n = \sum_s n_s$ is the total count.

From Equation 1 we see that the data log likelihood $\log P(\{\mathbf{y}\})$ can be computed solely from the transition counts $\{n_{y,y'}\}$ (i.e., the counts $\{n_{y,y'}\}$ form a set of sufficient statistics for this model). The transition counts $\{n_{y,y'}\}$ can be stored in an $M \times M$ matrix $\mathbf{C}$, a matrix that in many applications is highly sparse.

The reduced-state model can be viewed as a constrained hidden Markov model (HMM), with the constraint that each symbol y can appear in only one state $s(y)$, i.e., $P(y|s) = 0$ for all s except $s(y)$. While the Expectation-Maximization (EM) algorithm can in principle be used to train this constrained model, it is in general not a particularly effective approach since the solutions exist at "corners" of parameter space (i.e., in our solution we seek many $P(y|s) = 0$ terms, but do not know ahead of time which terms are zero and which are not). We will see later that assignment-style algorithms, that move symbols from state to state to maximize the likelihood (or posterior probability in a maximum a posterior (MAP) framework) are empirically much more effective than EM for this problem.

3 Related Work

Our SVD-based method is based on a block-constant matrix derived from the symbol transition counts. We show in the next section that the constrained HMM model leads to a matrix with block-constant structure. Once we have the block-structured matrix, we use spectral methods to find the clusters. This step is similar in spirit to previous work in image segmentation that uses the block structure of a pairwise similarity matrix to find clusters (Perona and Freeman, 1998; Weiss, 1999; Meilă and Shi, 2001).

For example, Meilă and Shi (2001) normalize the pairwise similarity matrix into a Markov-chain transition matrix, where the eigenvectors are piecewise-constant iff the matrix is "block-stochastic." In theory, we could also use their *Modified NCut* (MNCut) algorithm, since our normalized symbol transition matrix is also block-stochastic. But in our model we are dealing with an actual Markov chain; the symbol transition counts are not similarities as assumed in the MNCut algorithm. Furthermore, as we will see in Section 4, our Markov chain allows the derivation of a much stronger matrix structure, i.e., a block-constant matrix. This leads to a more robust algorithm in the face of limited amounts of training data, as we will show at the end of Section 4.

The search-based algorithms that we propose are closely related to bigram word clustering in language modeling. With word clustering, each word w belongs to a category $c(w)$. The bigram model predicts word w_t from the previous word w_{t-1} as follows:

$$P\big(w_t|w_{t-1}\big) = P\big(c(w_t)|c(w_{t-1})\big)P\big(w_t|c(w_t)\big).$$

Brown et al. (1992) initialize the word clusters with hierarchical clustering, then cycle through the words, moving each word to its best class. Martin et al. (1998) extend this approach to the trigram model; they initialize the clusters by putting each of the $K-1$ most frequent words into a separate class, and the remaining words into one class.

The greedy algorithm of cycling through the words and moving each word to its best class corresponds to the iterative conditional mode (ICM) style heuristic (Besag, 1986) that we later use in our search-based algorithms. We formulate the problem in the general framework of searching (namely to search for the best assignment of M symbols to K states, among all possible assignments) and investigate the effectiveness of a number of different heuristic search algorithms from artificial intelligence (AI) for this problem.

4 An SVD Formulation of the Problem

The SVD method is based on the $M \times M$ matrix defined by

$$H_{y,y'} \equiv P\big(s(y')|s(y)\big)/P\big(s(y')\big).$$

The above definition implies that an entry (y, y') in H depends only on $s(y)$ and $s(y')$ and not on the individual symbols y and y'. When the symbols in the same state are numbered consecutively, $\mathbf{H}$ will be a block-constant matrix with $K \times K$ blocks. For example, when $M = 4$, $K = 2$, $s(y = 1, 2) = 1$, $s(y = 3, 4) = 2$, $\mathbf{H}$ will be of the following form:

$$\begin{array}{c|cccc} \text{s} & 1 & 1 & 2 & 2 \\ \hline 1 & a & a & b & b \\ 1 & a & a & b & b \\ 2 & c & c & d & d \\ 2 & c & c & d & d. \end{array}$$

In general, $\mathbf{H}$ will be a *permuted* block-constant matrix, e.g., when $s(y = 1, 4) = 1$, $s(y = 2, 3) = 2$,

$$\begin{array}{c|cccc} \text{s} & 1 & 2 & 2 & 1 \\ \hline 1 & a & b & b & a \\ 2 & c & d & d & c \\ 2 & c & d & d & c \\ 1 & a & b & b & a. \end{array}$$

Let the SVD (singular-value decomposition) of $\mathbf{H}$ be

$$[\mathbf{u}_1\, \mathbf{u}_2\, \ldots\, \mathbf{u}_M]\, \mathrm{diag}(\sigma_1\, \ldots\, \sigma_K\, 0\, \ldots\, 0)\, [\mathbf{v}_1\, \mathbf{v}_2\, \ldots\, \mathbf{v}_M]\, ,$$

where $\mathbf{u}_1$, $\mathbf{u}_2$, …, $\mathbf{u}_M$ and $\mathbf{v}_1$, $\mathbf{v}_2$, …, $\mathbf{v}_M$ are the left and right singular vectors, respectively, and σ_1, …, σ_K, 0, …, 0, are the singular values. (Only the first K singular values are nonzero, as the rank of $\mathbf{H}$ is K.)

The elements in a singular vector are constant for the symbols in the same state. In other words, if $s(y) = s(y')$, then $\mathbf{u}_i[y] = \mathbf{u}_i[y']$, $\mathbf{v}_i[y] = \mathbf{v}_i[y']$. We represent each symbol y by the tuple

$$(\sigma_1\mathbf{u}_1[y],\, \ldots,\, \sigma_K\mathbf{u}_K[y],\, \sigma_1\mathbf{v}_1[y],\, \ldots,\, \sigma_K\mathbf{v}_K[y]), \quad (2)$$

which is the same for all symbols in a state. By looking at which symbols have the same tuples, we can correctly assign all the symbols to their states.

To summarize, in the hypothetical case where we know the exact $\mathbf{H}$, we can run SVD on $\mathbf{H}$, then set the number of states K to be the number of nonzero singular values, and assign the symbols with the tuples (as defined by Equation 2) to the same state.

In practice, however, we do not know $\mathbf{H}$ exactly (e.g., it is estimated from data). Note that

$$\begin{aligned} H_{y,y'} =& P\big(s(y')|s(y)\big)/P\big(s(y')\big) \\ =& \frac{P\big(s(y')|s(y)\big)P\big(y'|s(y')\big)}{P\big(y'|s(y')\big)P\big(s(y')\big)} \\ =& \frac{P\big(y'|y\big)}{P\big(y'|s(y')\big)P\big(s(y')\big)} \end{aligned}$$

can be empirically estimated by

$$\begin{aligned} B_{y,y'} \equiv& \frac{n_{y,y'}}{n_y} \frac{1}{n_{y'}/n_{s(y')}} \frac{1}{n_{s(y')}/n} \\ =& \frac{n_{y,y'}n}{n_y n_{y'}}. \end{aligned}$$

We can in practice still run SVD on $\mathbf{B}$ to assign the symbols to the states. However $\mathbf{B}$ will have more than K singular values. Thus, the tuples for the symbols in the same state will not be exactly the same. We can use a clustering algorithm (e.g., K-means) on the tuples to assign the symbols to the states.

The above algorithm (we call it "SVD-B") runs SVD on $\mathbf{B}$. It minimizes the sum of squared errors (SSE) of the model (i.e., the permuted block-constant matrix $\mathbf{H}$) with respect to $\mathbf{B}$. This effectively assumes that $B_{y,y'}$ has a Gaussian distribution with mean $H_{y,y'}$, and a common variance σ^2.

We can also run SVD on $\mathbf{C}$. Note that

$$\mathbf{B} = n\mathbf{W}^{-1}\mathbf{C}\mathbf{W}^{-1}$$

where $\mathbf{W}$ is a diagonal matrix, $W_{yy} = n_y^{-1}$. If we run SVD on $\mathbf{C}$ first, $\mathbf{C} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, we can still get $\mathbf{B}$:

$$\begin{aligned} \mathbf{B} &= n\mathbf{W}^{-1}\big(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\big)\mathbf{W}^{-1} \\ &= n(\mathbf{W}^{-1}\mathbf{U})\mathbf{\Sigma}(\mathbf{W}^{-1}\mathbf{V})^T \\ &= n\mathbf{P}\mathbf{\Sigma}\mathbf{Q}^T, \end{aligned}$$

where $\mathbf{P} = \mathbf{W}^{-1}\mathbf{U}$, and $\mathbf{Q} = \mathbf{W}^{-1}\mathbf{V}$. The rows of $\mathbf{P}$, $\mathbf{Q}$ are approximately constant for the symbols in the same states. We replace the singular vectors in Equation 2 by the columns of $\mathbf{P}$, $\mathbf{Q}$, and then run the clustering algorithm to put the symbols into states. We call this version of the algorithm "SVD-C".

Experimental Results on Simulated Data

To evaluate the above two SVD-based algorithms (SVD-B and SVD-C), we ran them on simulated data,

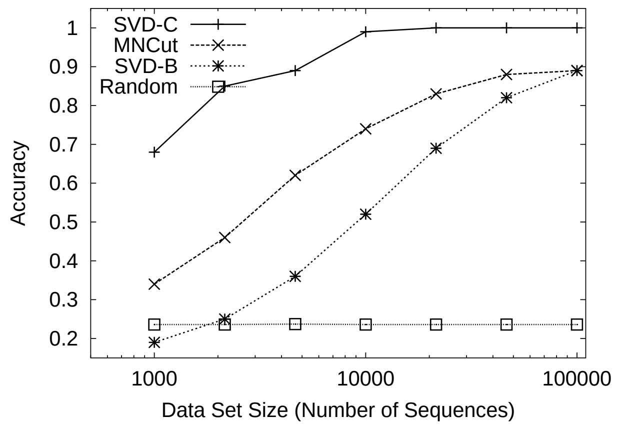

Figure 1: Accuracy of the SVD methods. SVD-C: SVD on the matrix **C** of symbol transition counts. MNCut: *Modified NCut* algorithm (Meilă and Shi, 2001). SVD-B: SVD on the permuted block-constant matrix **B**. Random: random assignment (as a baseline method).

together with a baseline algorithm that randomly assigns the M symbols to the K states.

When generating the simulated data, the number of symbols was $M{=}100$, the number of states was $K{=}10$, and each state had $M/K = 10$ symbols. In each experiment, the model parameters $\boldsymbol{\theta} = \{A, \{P_{y|s(y)}\}\}$ were set randomly, and N sequences were generated, each with length between 5 and 10. From the N sequences, the symbol transition counts were recorded in the matrix **C**. We then ran the 3 clustering algorithms, and computed their accuracy against the "true" clustering. To compute the accuracy of a clustering algorithm, we ran a maximum weight bipartite matching algorithm (Goldberg and Kennedy, 1995) to find the best matching between the computed clusters and the true clusters.

In Figure 1, we show the accuracy of the algorithms on simulated data. Each point in the plot shows the median accuracy from 1000 experiments. (Recall that $\boldsymbol{\theta}$ is different for each experiment.) The "Random" points correspond to a baseline method of randomly assigning the M symbols to K states. The number of symbol sequences, N, is varied from 1000 to 100000. When $N{=}1000$, the data are pretty sparse: of the 10000 elements of the matrix **C**, about 23% are 0, another 22% are 1 or 2, and the median is 3. In this case, the SVD-B algorithm breaks down (it is even worse than the baseline random-assignment algorithm), while the SVD-C algorithm is fairly accurate. With larger N, i.e., more data, the accuracy of both SVD-B and SVD-C increases.

From Figure 1, we can see that SVD-C is consistently better than SVD-B. This tells us that it is better to approximate **C** than **B**. This should be expected as as the data log likelihood is directly related to the matrix **C** of symbol transition counts.

In Figure 1, we also show the accuracy of running the *Modified NCut* (MNCut) algorithm (Meilă and Shi, 2001), which is based on clustering the rows of the first K eigenvectors of $\mathbf{W}^{-1}\mathbf{C}$. MNCut is not as good as SVD-C for our clustering problem, although it is better than SVD-B.

5 Search-based Algorithms

An alternative viewpoint from the SVD approach is to view the assignment of symbols to states as a combinatorial search problem: searching the space of all possible assignments for the "best" assignment. The score function being optimized is the data log likelihood in Equation 1.

Heuristic Search Algorithms

We experimented with a number of greedy local search algorithms, such as variants of GSAT (Selman et al., 1992) and ICM (Besag, 1986). These algorithms all start from a (random) initial cluster assignment, and improve the solution via local search steps, until a local maximum of the score function is achieved. The algorithms can be re-started at different randomly chosen starting points a number of times.

The main difference between the various algorithms is the heuristic used to choose a symbol to move from one state to another. We experimented with the following heuristics:

- GSAT: Of the $N \times (K-1)$ possible moves choose the one that leads to the largest increase in the log-likelihood.
- Similar to GSAT, but only 10% of the $N \times (K-1)$ possible moves are sampled. The best move is then executed.
- ICM: Go through the symbols sequentially, moving each symbol to the state that maximizes the log likelihood.
- Similar to ICM, except that the order of scanning the symbols is randomized for different passes.

In addition to the above greedy local search algorithms, we also tried simulated annealing as a global search algorithm. The temperature T starts at a high level T_0, and then falls gradually as $T \leftarrow 0.9T$. At

Algorithm	Accuracy	Time/ICM
GSAT	0.889	16.8
GSAT with 10% sampling	0.881	3.3
ICM	0.885	1.0
ICM with randomized order	0.895	1.1
Simulated Annealing	0.959	316.3
SVD-C	0.681	18.1
Unconstrained HMM	0.43	1464.6
Random Assignment	0.234	-

Table 1: Accuracy and computation times of various algorithms for assigning symbols to states. with $N = 1000$ sequences, $M = 100$, $K = 10$. "Time/ICM" is ratio of computation time to that of ICM.

each temperature level T, `n_try` tries are attempted. At each try we uniformly sample one possible move (out of the total of $N \times (K-1)$). The move is executed according the following probability:

$$P(\text{move}) = \begin{cases} 1 & \text{if } \Delta E < 0, \\ e^{-\Delta E/T} & \text{if } \Delta E \geq 0, \end{cases}$$

where ΔE is the decrease in data log likelihood.

Efficiently Computing the Change in Log-likelihood

Each of these algorithms requires the efficient computation of the change in data log likelihood when moving a symbol from one state to another. The data log likelihood can be computed using just the counts:

$$\log P(\{\mathbf{y}\}) = \left[\sum_{s,s'} n_{s,s'} \log n_{s,s'}\right] + \left[\sum_{y} n_y \log n_y\right] - 2 \times \left[\sum_{s} n_s \log n_s\right]. \quad (3)$$

There is no need to refer to the model parameters $\boldsymbol{\theta} = \{A, \{P_{y|s(y)}\}\}$ explicitly. We only need to maintain the counts $\{n_s, n_{s,s'}\}$, and when moving a symbol, compute the change in Equation 3.

Experimental Results on Simulated Data

To evaluate the various search-based algorithms described about we ran the algorithms on simulated data. The experimental settings are the same as for the SVD-based algorithms described earlier. In Table 1 we show the accuracy and computation time of the algorithms when the number of sequences is $N = 1000$.

From the table, we can see that the best accuracy is achieved by simulated annealing, but it is also the slowest. The second most accurate algorithm is ICM (in particular the version with randomized scanning order.) ICM is also the fastest algorithm. The "10% sampling" version of GSAT greatly reduces the running time of GSAT, with relatively little decrease in accuracy. Note that alternative clustering approaches, such as an unconstrained HMM trained using EM, are generally far less effective than the heuristic search methods, as they do not take advantage of the fact that each symbol belongs to only one state.

All of the search algorithms are more accurate than the SVD-C algorithm indicating that it is beneficial to directly optimize the data likelihood for this problem.

6 Application to User Modeling

To illustrate the applicability of these ideas for real-world data sets we tested the ICM algorithm on two applications. In the first application we ran the algorithms on the Purdue UNIX user data, available from the UCI KDD Archive (Hettich and Bay, 1999) and originally reported in Lane (1999). The symbols are various UNIX commands such as '`ls`', '`cd`', '`netscape`', '`gcc`', etc, with approximately $M = 400$ symbols in total, obtained over several months of logged command line usage data from 12 different users. Each sequence is a session (from login to logout) of a user. Being able to learn a "user model" of each individual's sequential behavior has a number of different potential applications, such as intrusion detection and online prediction and personalization.

We ran the ICM algorithm on each user's sequences and found a variety of meaningful patterns. In Figures 2–3, we show the state diagrams for two users, USER2, USER3. Only the major state transitions (corresponding to the top 20% largest entries in the transition matrix) are shown. For each state, we show the top 3 symbols (with the largest observation probabilities $P(y|s)$).

From these figures, we can see that related commands are often grouped together. For example, state 7 of USER2 (Figure 2) contains the symbols '`a.out`', '`xcc`', '`dbx`' which are related to compiling, debugging and running C programs. In the same figure, state 3 is related to LaTeX.

In Figure 3, we find a clique of 3 states: states 9, 3, and 4 represent a typical "edit-compile-execute" pattern.

7 Application to Word Segmentation

In language modeling there are numerous applications of Markov models where the symbols consist of individual words. As an example, in Chinese text, there is

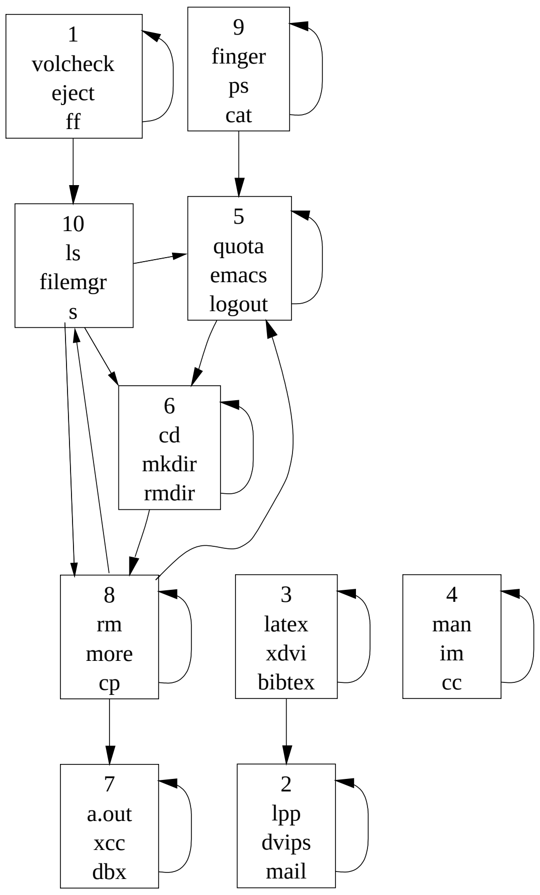

Figure 2: Major state transitions for USER2.

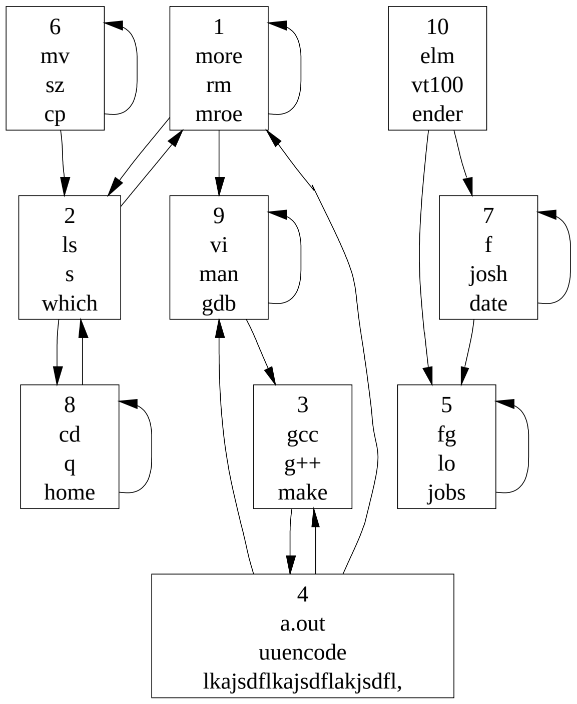

Figure 3: Major state transitions for USER3.

no space between words and word segmentation algorithms are used to segment a sentence into words (e.g., for Chinese information retrieval (Nie et al., 2000)). Word segmentation can also be applied to biological sequences (Bussemaker et al., 2000) in an attempt to "parse" protein sequences into component "words". In what follows we illustrate the use of Markov models for word segmentation using English as an example language, e.g.,

"Thisisadog" $\rightarrow$ **"This is a dog"**.

We use English since there is in effect an infinite amount of segmented training data available for algorithm evaluation.

A popular probabilistic model that works reasonably well for word segmentation is the "word-independence" model. Under this model, the probability of a sentence is simply the product of the probabilities of the words in the sentence:

$$P(w_1 w_2 \ldots w_m) = \prod_{i=1}^{m} P(w_i),$$

where the w_i's represent the words. The model parameters consist of a list of words and their probabilities. Given a sentence (as a string of characters), a sentence is segmented into w_1, w_2, $\ldots$, w_k such that $\prod_i P(w_i)$ is maximized. This can be easily done using dynamic programming (Ponte and Croft, 1996).

Although the independence model works quite well for many sentences, it does not correctly parse certain word combinations where word interdependence provides vital clues to interpretation, e.g., "`a dead he at`" versus "`a dead heat`".

Here we propose to model the words using the type of constrained HMM defined earlier in the paper. Specifically, we assume that each word belongs to one of K word classes, and the sequence of word classes in a sentence forms a Markov chain. Note that building a Markov chain directly on the words is impractical, since M is typically $O(10^5)$. Words correspond to symbols in the model, word classes to states. The maximum likelihood segmentation of a sentence into words can also be done using dynamic programming.

Our experiments were carried out a 92M-byte corpus of the 227 files in the "Classic" subdirectory of the *Wiretap Online Library* (`http://wiretap.area.com/Gopher/Library/`). The number of distinct words is $M = 140072$.

To estimate the parameters for the word independence model, we count the occurrences of each word in the corpus. To estimate the parameters for the constrained HMM, we count the transitions between each

```
WI:  it old john it was you
HMM: i told john it was you

WI:  the you that my side corrected me
HMM: the youth at my side corrected me

WI:  i a man american
HMM: i am an american

WI:  what hotel are you stopping a there
HMM: what hotel are you stopping at here

WI:  which ended in a dead he at
HMM: which ended in a dead heat

WI:  the eldest being as on
HMM: the eldest being a son
```

Table 2: Some examples where the constrained HMM corrects the error of the word independence model. `WI`: word independence model. `HMM`: constrained HMM.

pair of words, then run the clustering algorithm to put the words into $K = 48$ word classes.

After both the HMM and independence models are trained, we run the maximum likelihood word segmentation algorithm on the unsegmented sentences in the corpus (i.e., spaces in the sentences have been removed).

The error rate (the number of missed words divided by the total number of words in the corpus) of the word independence model is 1.55%. The error rate of the constrained HMM is significantly lower at 0.79%.

In Table 2 we show some examples where the constrained HMM makes the correct segmentation while the word independence model does not. In the first example, the word independence model segments "`itoldjohn`" into "`it old john`", as it only considers the marginal probabilities of the words. The constrained HMM avoids this segmentation because it also considers the transition probabilities among the word classes. This is illustrated in Figure 4, where we show the states for the words with the log transition probabilities. For each state, we also list its top words, i.e., those with highest $P(\text{word}|\text{state})$. We can see that it is very unlikely to go from "`it`" to "`old`". For another example, see Figure 5. From these two examples, we can also see that the word classes found by the clustering algorithm are quite meaningful. See also Table 3 for some other word classes.

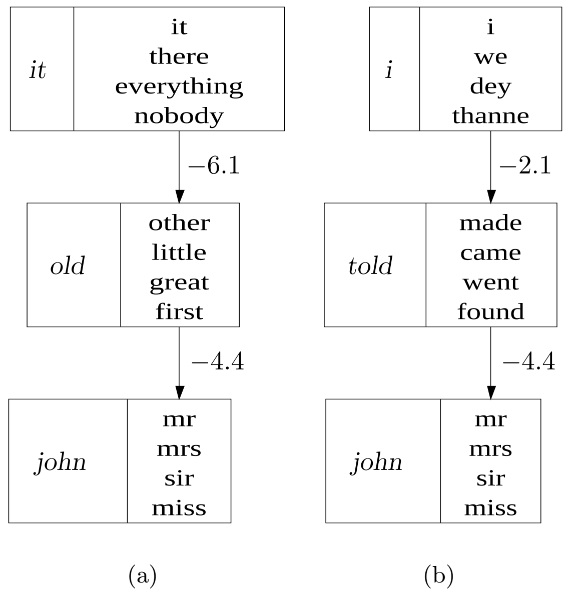

Figure 4: State transitions for (a) $\boxed{\texttt{it old john}}$ and (b) $\boxed{\texttt{i told john}}$. The numbers are log transition probabilities.

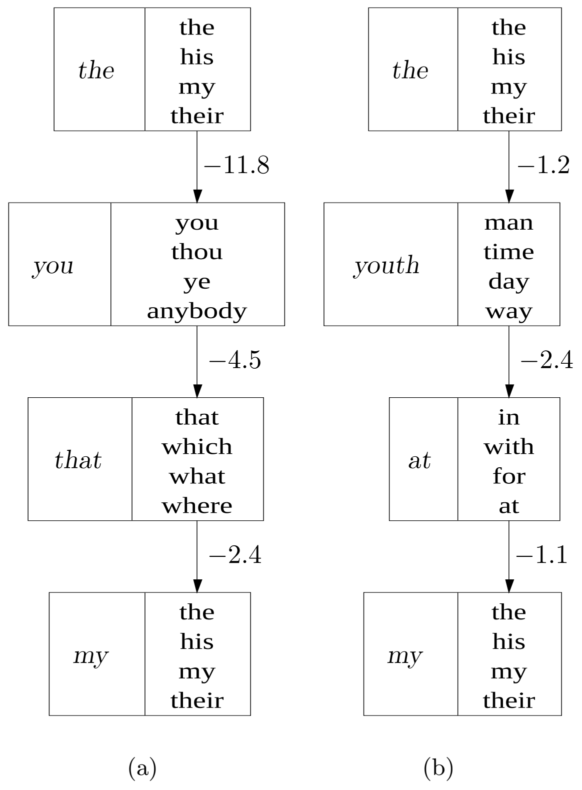

Figure 5: State transitions for (a) $\boxed{\texttt{the you that my}}$ and (b) $\boxed{\texttt{the youth at my}}$. The numbers are log transition probabilities.

god	would	said	come
life	will	asked	go
night	could	cried	look
death	can	replied	love
course	did	says	use
water	should	answered	help
others	must	returned	live
nature	may	continued	speak
money	shall	added	turn
truth	might	laughed	care
fact	cannot	exclaimed	answer
reason	shalt	declared	talk
him	good	men	place
me	long	people	work
them	better	things	side
us	high	years	light
himself	true	words	part
home	dead	days	power
myself	large	children	end
herself	certain	feet	state
themselves	short	women	matter
thee	dark	times	kind
itself	hard	friends	point
yourself	strong	hours	spirit

Table 3: Some word classes found by the ICM-based symbol clustering algorithm.

8 Conclusions

Reduced-state Markov chains can provide useful tools for modeling sequential data with large symbol alphabets. The general idea of equivalence classes of Markov states has been known and utilized in the language modeling literature for some time. In this paper we explored a number of new algorithms in this context, demonstrated a theoretical link to singular value decomposition, and illustrated how these ideas can be applied to applications in user modeling and word segmentation. An interesting direction for future work lies in learning reduced-state models via Bayesian hierarchical modeling, where (for example) the M sets of transition rows in the matrix $\mathbf{C}$ could be modeled as noisy observations from a mixture of K Dirichlet "priors" whose parameters can be learned from the data.

Acknowledgements

This work was supported by research grants from the National Science Foundation (awards IRI-9703120 and IIS-0083489), the Jet Propulsion Laboratory, and Microsoft Research. Xianping Ge was supported by an IBM Graduate Fellowship.

References

J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society, Series B (Methodological)*, 48(3):259–302, 1986.

P. F. Brown, V. J. D. Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18 (4):467–479, 1992.

H. J. Bussemaker, H. Li, and E. D. Siggia. Regulatory element detection using a probabilistic segmentation model. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB 2000)*, 2000.

A. V. Goldberg and R. Kennedy. An efficient cost scaling algorithm for the assignment problem. *Mathematical Programming*, 71:153–177, 1995.

S. Hettich and S. D. Bay. The UCI KDD Archive [http://kdd.ics.uci.edu], 1999. Irvine, CA: University of California, Department of Information and Computer Science.

T. Lane. Hidden Markov models for human/computer interface modeling. In *Proceedings of the IJCAI-99 Workshop on Learning about Users*, pages 35–44, 1999.

S. Martin, J. Liermann, and H. Ney. Algorithms for bigram and trigram word clustering. *Speech Communications*, 24:19–37, 1998.

M. Meilă and J. Shi. A random walks view of spectral segmentation. In *Eighth International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2001.

J.-Y. Nie, J. Gao, J. Zhang, and M. Zhou. On the use of words and n-grams for Chinese information retrieval. In *Fifth International Workshop on Information Retrieval with Asian Languages*, Sept. 30–Oct. 1 2000.

P. Perona and W. T. Freeman. A factorization approach to grouping. In *European Conference on Computer Vision*, 1998.

J. M. Ponte and W. B. Croft. USeg: A retargetable word segmentation procedure for information retrieval. In *Symposium on document analysis and information retrieval (SDAIR '96)*, 1996.

B. Selman, H. J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, 1992.

Y. Weiss. Segmentation using eigenvectors: A unifying view. In *Proceedings IEEE International Conference on Computer Vision*, pages 975–982, 1999.