X-mHMM: An Efficient Algorithm for Training Mixtures of HMMs when the Number of Mixtures is Unknown

Zoltán Szamonek[1,2] and Csaba Szepesvári[1]
[1]Computer and Automation Research Institute of the Hungarian Academy of Sciences
Kende u. 13-17, Budapest 1111, Hungary
[2]Eötvös University Budapest, Pázmány P. sétány 1/c, Budapest 1117, Hungary
zszami@elte.hu, szcsaba@sztaki.hu

Abstract

In this paper we consider sequence clustering problems and propose an algorithm for the estimation of the number of clusters based on the X-means algorithm. The sequences are modeled using mixtures of Hidden Markov Models. By means of experiments with synthetic data we analyze the proposed algorithm. This algorithm proved to be both computationally efficient and capable of providing accurate estimates of the number of clusters. Some results of experiments with real-world web-log data are also given.

1 Introduction

Clustering is the process of discovering natural groups in data. Applications of clustering include biology, chemistry, physics, computer science, image processing, sociology and finance. The data to be clustered can be either fixed length, finite-dimensional vectors or of varying length sequences. Clustering vectorial data has a vast literature (e.g. [2, 4]).

Sequential data clustering (SDC) is a relatively recent topic. Following Bicego and Murino [6], methods of SDC can be classified as belonging to one of the following three categories: *proximity-based* methods, *feature-based* methods and *model-based* methods. In proximity-based methods the algorithms' aim is to find appropriate distance or similarity measures and transform the problem into one that can be solved by any standard distance-based method (e.g. by agglomerative clustering). Feature-based algorithms first transform the variable-length sequences into feature vectors with fixed length, for which any standard clustering method that accepts vector-space data can be used. Model-based methods assume that for each cluster there exists a separate model responsible for generating the observations belonging to the given cluster. Within a probabilistic framework, assuming that sequence indexes are generated according to some prior distribution independently of each other, the compound model is called a mixture-model. The problem of finding the 'hidden' indexes of sequences is called latent variable modeling and is often solved using variants of the Expectation Maximization (EM) algorithm. One common property of these models is that if the indexes were given, the models could be trained by standard methods.

The sequence generating models can take many form, such as time-series models (e.g. AR, ARMA, GARCH, etc.), spectral models, observable operator models, or Hidden Markov Models (HMMs). In this paper we restrict ourselves to the case when the sequences are generated by HMMs where the observations take on values in a finite space, though we note in passing that our algorithm can be extended naturally to other mixture models, as well.

Recently, the literature on probabilistic mixture models has grown very quickly. Due to space limits we review here only a few of the most relevant works on mixture of HMMs (mHMMs). The earliest approaches of using HMMs for SDC are mostly related to speech recognition (e.g. [8]). These methods are mostly proximity-based. The first application of mHMMs in SDC is [9] where the author proposed the use of Baum-Welch training preceded by a clever initialization procedure; the number of clusters was found by an exhaustive search by selecting the number that yields the best Monte-Carlo cross-validated log-likelihood for the data. Li and Biswas [5] proposed to use the Bayesian Information Criterion (BIC) and the so-called Cheeseman-Stutz approximation to the posterior of models for the simultaneous estimation of the number of clusters and the number of states of the underlying HMMs [5]. In their algorithm, in each iteration a new HMM is constructed and is added to the set of mixtures. The algorithm stops when the partition posterior probability (which is estimated by a penalized log-likelihood of the data) starts decreasing. The above papers try to estimate the number of clusters by adding one model at a time followed by training on the whole data set and then

by testing the new mixture model by evaluating some score function. More recently, hierarchical agglomerative clustering techniques were proposed in [11]. In this approach, clustering starts with finer granularity than ultimately required, followed by merging 'neighboring' clusters in an iterative fashion. In this paper, however, no attempt was made to determine the number of clusters in an automatic way. The paper by Ypma and Heskes is particularly relevant for us as they considered the problem of mining weblogs, which is also one of our test domains [10]. Again, the problem of automatic estimation of the number of clusters was not considered in this paper.

All the papers that we are aware of follow either a naive strategy whereby in each iteration a different cluster number is tested for which a mixture model is trained typically 'from scratch', or they follow an *incremental* strategy where a small (monotonous) change is made to the number of clusters (the structure) and then the mixture model is retrained on the whole data set. The problem with the former *linear search* approach is that it throws away potentially useful information when training the new model with a higher number of mixtures, whilst the problem with the latter approach is that the selection of *which* model to merge/split can be very expensive relative to that after the search for this model only a single change is made to the final structure. Inspired by the X-means algorithm [7], where k-means clustering was extended so that the number of clusters is automatically estimated during the course of the clustering process, here we propose an algorithm that considers splitting *every* model in each iteration and decides about which models to split based on subsets of data associated with the individual models. This way the decision about which models to split becomes efficient since only a subset of the whole data is used in each of these tests. Further, since in each iteration many of the models can be split *simultaneously*, the algorithm potentially makes the search time for the number of clusters logarithmic in K^*, the optimal number of clusters. Our results with synthetic data show that the proposed algorithm is not only efficient but also yields accurate estimates of the number of clusters.

The paper is organized as follows: background on HMMs and mixture of HMMs are given in Section 2, X-mHMM, the proposed algorithm is given in Section 3, whilst experiments with synthetic and real-world data are given in Section 4. Conclusions are drawn in Section 5.

2 Background

2.1 Hidden Markov Models

Let m denote the number of symbols in a finite alphabet. For simplicity the members of this alphabet will be identified with the numbers $1, \ldots, m$. A standard discrete state, discrete observation HMM defines a probability distribution over the space of all observation sequences. Denoting by n the number of states, such a HMM can be defined by a triplet $\lambda = (\pi, A, B)$, where $\pi \in [0,1]^n$ is the initial distribution of states, $A = (a_{ij}) \in [0,1]^{n \times n}$ is the transition probability matrix and $B = (b_{ik})$ are the observation probabilities. Here a_{ij} is the probability that the next state is j assuming the current state is i ($1 \leq i, j \leq n$), whilst b_{ik} is the probability of observing symbol k when the state is i ($1 \leq i \leq n, 1 \leq k \leq m$).

The most well-known algorithm for training HMMs based on a number of observation sequences is the Baum-Welch algorithm [1]. For a sequence of observations $Y = (Y_1, Y_2, \ldots, Y_T)$ define $\xi_t(i, j|Y, \lambda) = P(X_t = i, X_{t+1} = j|Y, \lambda)$ and $\gamma_t(i|Y, \lambda) = \sum_{j=1}^{n} \xi_t(i, j|Y, \lambda)$. Then, given a sequence of observations $Y^{(1)}, \ldots, Y^{(N)}$, the Baum-Welch formulas for reestimating the transition and observation probabilities given a current set of parameters λ are

$$
\begin{aligned}
a'_{ij} &= \frac{\sum_{l=1}^{N} \sum_{t=1}^{T^{(l)}} \xi_t(i, j|Y^{(l)}, \lambda)}{\sum_{l=1}^{N} \sum_{t=1}^{T^{(l)}} \gamma_t(i|Y^{(l)}, \lambda)}, \\
b'_{ik} &= \frac{\sum_{l=1}^{N} \sum_{t=1}^{T^{(l)}} \gamma_t(i|Y^{(l)}, \lambda) \mathbb{I}(Y_t^{(l)} = k)}{\sum_{l=1}^{N} \sum_{t=1}^{T^{(l)}} \gamma_t(i|Y^{(l)}, \lambda)}.
\end{aligned}
$$

Here $\mathbb{I}(L) = 1$ iff $L = \text{true}$, and $\mathbb{I}(L) = 0$, otherwise. The Baum-Welch algorithm is an instance of the Expectation Maximization (EM) algorithm and as such in each iteration it increases the likelihood of the observed data.

In practice, the Baum-Welch algorithm is computationally expensive and is commonly replaced by *Viterbi training* (VT). VT replaces the computationally costly expectation step of EM by a step that is computationally less intensive. The price to be paid for the gain in computational efficiency is often the loss of accuracy: VT does not necessarily increase the likelihood of data in every step. VT works by finding the best alignment of the training sequences to the internal states of the model. For a given observation Y and model parameter λ, let $\hat{X}_t(Y; \lambda)$ be the state assigned to the tth observation of the sequence Y. Then, VT training is obtained if in the above formulas ξ_t is replaced by $\xi_t(i, j|Y, \lambda) = \mathbb{I}(\hat{X}_t(Y; \lambda) = i, \hat{X}_{t+1}(Y; \lambda) = j)$. In the context of speech recognition this algorithm has been described by Juang and Rabiner [3].

2.2 Mixtures of HMMs

We consider the following problem: assume that we are given a data set D of N observations $Y^{(1)}, \ldots, Y^{(N)}$. Each observation $Y^{(i)}$ consists of a sequence of symbols $Y_1^{(i)}, \ldots, Y_{L_i}^{(i)}$ of varying length. The problem of clustering sequences is to discover a natural grouping of the sequences into some clusters based on the observed data D.

Assuming that the number of clusters is given and is denoted by K, one natural probabilistic model for this problem takes the form of a finite mixture:

$$f_K(Y) = \sum_{j=1}^{K} p_j f_j(Y|\lambda_j).$$

Here Y denotes a sequence, p_j is the prior probability of the jth model and $f_j(Y|\lambda_j)$ is the probability of observing Y given that the parameters of the jth model are determined by λ_j. In this paper we shall assume that the models are given by HMMs, hence $\lambda_j = (\pi_j, A_j, B_j)$ are the parameters of the jth HMM.

Note that a mixture of HMMs given by $p = (p_1, \ldots, p_K)$ and $\Lambda = (\lambda_1, \ldots, \lambda_K)$ can be viewed as a single HMM with a state space of size $n = n_1 + \ldots + n_K$, where n_j is the size of the state space of the jth model [9]. The transition probability matrix of this composite HMM is $A = \mathrm{diag}(A_1, \ldots, A_K)$, the observation matrix is $B = \mathrm{diag}(B_1, \ldots, B_K)$, whilst the initial state distribution π is given by $(p_1\pi_1^T, \ldots, p_K\pi_K^T)^T$. Hence, in theory, the Baum-Welch algorithm can be used to find the parameters of the mixture models if the model parameters are initialized to respect the special structure of this composite HMM model, i.e., by fixing a_{ij} to zero whenever i and j do not belong to the same submodel. This follows because then for such pairs $\xi_t(i, j|Y, \lambda)$ will always be zero and hence these parameters remain zero forever. The same holds for the observation parameters.

Viterbi training (VT) can be generalized to this case in the obvious way. However, as discussed before, although VT speeds up training, it can also result in a loss of accuracy. In order to mitigate this effect, we adopted an intermediate option in which each sequence is assigned to the model that is the most likely to generate it. The connection to VT should be clear: if one introduces the hidden variable I denoting the index of the model that generated the sequence Y as a non-emitting state then the procedure can be thought of as the partial Viterbi alignment of Y to the states where only the alignment w.r.t. I is considered. This procedure can be thought of as the natural extension of K-means clustering to sequence data and has been proposed in many of the papers dealing with training mixture of HMMs. This method, that we call semi-VT, is used in the algorithm proposed below.

3 The X-mHMM Algorithm

X-mHMM applies the idea underlying X-means to find the number of mixtures in a mixture of HMM model. The input to the algorithm is the data set D containing the observed sequences, an upper bound on the number of mixtures ($K_{\max}$), and a natural number s that is used in the stopping condition and whose role will become clear later. The main structure of the algorithm is the same as that of X-means: in each iteration of the main loop all models are considered as split candidates. Once splits are done, the full model is retrained on the whole data set. The pseudocode of the algorithm is listed in Figure 1, whilst the pseudocode of split-attempts is listed in Figure 2.

1: **function** XmHMM $(D, K_{\max}, s)$
2: m:=[λ_1]; $p := [p_1]$; K:=1;
3: $t := 0$; $l := 1$;
4: train([p,m], D);
5: **while** $((K \leq K_{\max})\&\&(t \leq l + s))$ **do**
6: $t := t + 1$;
7: m' := [ ]; p' := [ ]; *// lists of new models*
8: **for** $i \in \{1, .., K\}$ **do**
9: $D^{(i)} := \{Y \in D | f(Y|\lambda_i) > \max_{k \neq i} f(Y|\lambda_k)\}$;
10: (l, m', p'):=splitAttempt($m', p', l, D^{(i)}, p_i, \lambda_i$);
11: **end for**
12: $m := m'$; $p := p'$; K := size(m);
13: **if** $(l = t)$ **then**
14: train([p,m], D);
15: **end if**
16: **end while**
17: **return** (p,m).

Figure 1. Algorithm X-mHMM

Let us explain now some details of the algorithm. The algorithm maintains a list of models $m = [\lambda_1, \ldots, \lambda_K]$ and a list of respective prior probabilities $p = [p_1, \ldots, p_K]$. These lists are initialized by training a single HMM on the whole data set (the method 'train' takes its first argument by reference, i.e., it changes them when it returns). In each step of the main loop, the data is partitioned into disjoint parts (see line 9) and each model in list m is considered as a candidate of splitting. The splitting decision is made in function 'splitAttempt'.

3.1 Split attempts

Function 'splitAttempt' takes a subset of the full data set, a model λ and its prior p. It then creates two perturbed models (we will shortly discuss this step below) that will serve as the initialization for the 'split' models. It is then decided (Figure 2, line 3) if the data is better modelled by a mixture of 2 HMMs than with a single HMM. In theory, given a prior over the models and model parameters, the probabilities of the two alternatives (i.e., $K = 1$, $K = 2$) should be used to decide which of the alternatives to choose. Assuming flat priors over the parameters, the respective probabilities reduce to the expected log-likelihood of the data given K (the models are integrated out). The expected log-likelihood can be estimated in a number of dif-

1: **function** splitAttempt(m', p', l, D, p, λ)
2: $[\hat{\lambda}_1, \hat{\lambda}_2]$:= perturb(λ);
3: **if** (cvScore($\hat{\lambda}_1, \hat{\lambda}_2$, D) > score([p, λ], D)) **then**
4: score0 := train([(q_1, q_2),($\hat{\lambda}_1, \hat{\lambda}_2$)], D);
5: $q'_1 := 1$; $q'_2 := 1$; $\hat{\lambda}'_1 := \hat{\lambda}_1$; $\hat{\lambda}'_2 := \hat{\lambda}_2$;
6: score1 := train([q'_1,$\hat{\lambda}'_1$], D);
7: score2 := train([q'_2,$\hat{\lambda}'_2$], D);
8: $l := t$;
9: **if** (score1>score2 && score1>score0) **then**
10: append(m', $\hat{\lambda}'_1$); append(p', p);
11: **else if** (score2>score1 && score2>score0) **then**
12: append(m', $\hat{\lambda}'_2$); append(p', p);
13: **else**
14: append(m', $[\hat{\lambda}_1, \hat{\lambda}_2]$);append(p', [$p * q_1$,$p * q_2$]);
15: **end if**
16: **else**
17: append(m', λ); append(p', p);
18: **end if**
19: **return** (l,m',p').

Figure 2. Function 'splitAttempt'

ferent ways. Following [7, 5], we first attempted to use BIC that uses a Laplacian approximation to the model-posterior, as it worked well in the cited works and is much cheaper than its alternatives such as e.g. cross-validation. However, our initial experiments showed that (at least for our data) the decisions based on BIC can be very sensitive to the number of training samples. In particular, we found that when the number of samples is small then splitting often stops prematurely. Since, during the course of iterative splittings, the number of samples per model becomes small as more models are added, we opted for cross-validation, which is more expensive but according to our experience yields more reliable estimates. We employed simple n-fold cross-validation where the data is partitioned into n disjoint parts.[1] The 'cvScore' function returns the corresponding estimated log-likelihood of the data. The returned score is compared with the score of the original model (λ) evaluated on the input data of 'splitAttempt'. This comparison favors the single-model alternative since the data used for calculating the score of model λ is used in its training. However, empirically we have found that the bias is weak and can be sufficiently counterbalanced by choosing the number of cross-validation partitions to be small (in our experiments we used $n = 3$). Using a small number of partitions introduces unwanted randomness in the decision process, but this effect can be mitigated by an appropriate stopping condition in the main loop of the algorithm. If the comparison favors the mixture model, then a 2-mixture HMM is trained to obtain the mixture model (q_1, q_2), $(\hat{\lambda}_1, \hat{\lambda}_2)$. It is well known that HMM training can become stuck in local minima. Actually, it can happen that the comparison favors the 2-mixture HMM only because the single model λ is suboptimal and starting the (mixture) training process from the perturbed models yields one model that models the whole data better than the original model, whilst the other model becomes 'dead' (only a small fraction of the samples is assigned to it). Therefore, we have added tests to check if this were the case. In particular, we have decided to compare the scores of fully trained versions of the models $\hat{\lambda}_1$, $\hat{\lambda}_2$ (lines 6–7) on the data and to test if any of these models yields a better score than the score of the 2-mixture HMM. Here, scores are calculated on the training data in all cases. This creates some bias towards the mixture model. However, we can argue that this bias is not harmful as it agrees well with the previous decision that mixture model should be preferred. Further, this decision was biased towards the single HMM variant. The comparison here gives a 'last chance' for rejecting the mixture hypothesis. The function returns the adjusted lists, as well as the variable l that indicates the last time that some models were changed when attempting the splits.

[1] In [9], Monte-Carlo cross-validation was found to yield more reliable estimates. Our experience suggests that the simple n-fold cross-validation works equally well, at least on the data sets that we used.

3.2 Main loop of X-mHMM and stopping criterion

The loop through the list of models in X-mHMM continues until all models are subject to the split-tests and the optional changes. At the end, the list of models, m, and the list of priors, p, are updated with the respective new lists, m', p' and the number of mixtures K is adjusted. When any of the models is changed then the new mixture model is retrained on the whole data set. Note that this process usually stops after a very small number of iterations if only a few changes were made to the models. The main loop of X-mHMM exits when the number of clusters found exceeds $K_{\max}$, the *a priori* upper bound on the number of clusters, or when for s consecutive steps no changes were made to the models. By increasing s one can mitigate the influence of the randomness introduced in the split-tests. The method returns the mixture of HMM models found.

3.3 Model perturbation

One detail not given earlier is how a model is perturbed to obtain two new models in function 'perturb'. The goal here is to obtain sufficiently distinct models. This is currently implemented by the following straightforward procedure: consider first the perturbation of the state transition matrix. For each pair of state indexes, i, j, an unbiased Bernoulli random variable V_{ij} taking on the values

$0, 1$ is sampled. If $V_{ij} = 0$ then we let $\tilde{a}_{ij}^{(1)} = 2a_{ij}$ and $\tilde{a}_{ij}^{(2)} = a_{ij}/2$, whilst if $V_{ij} = 1$ then we let $\tilde{a}_{ij}^{(1)} = a_{ij}/2$ and $\tilde{a}_{ij}^{(2)} = 2a_{ij}$. The transition matrix for the model $\hat{\lambda}_1$ (resp. $\hat{\lambda}_2$) is obtained by normalizing $\tilde{a}_{ij}^{(1)}$ (respectively, $\tilde{a}_{ij}^{(2)}$) appropriately so as to ensure that the normalized matrix is a stochastic matrix. The same procedure is applied to both the initial state distribution and the observation probability matrix.

This method, albeit quite simple, achieves sufficient distinctness of the perturbed models, while maintaining their structure (for example zero transitions are kept). Note that semi-VT also helps to make the models less similar to each other. Making the models less similar to each other is crucial in the splitting process: if the HMMs of the 2-mixture HMM were similar to each other then it is likely that one of them is sufficient for modelling the partition. VT is not that crucial in the main part of X-mHMM (line 14) where one could also use the Baum-Welch procedure. We note in passing that it would be advantageous to use semi-VT when the mixtures are "overlapping".

4 Experiments

We have experimented with both synthetic and a real-world data. The latter data was gathered from a popular web-server's log, the task being to cluster users by their browsing behaviour.

The purpose of the experiments with the synthetic data was to study the behaviour of the algorithm in a controlled environment. In particular, we were interested in the algorithm's scaling and robustness properties, as well as its accuracy, in comparison to the performance of a naive algorithm. We call this algorithm the linear-search algorithm and abbreviate it to *lis-mHMM*. This algorithm searches for the number of clusters by training increasingly larger mixture models, the number of mixtures being increased by one in consecutive iterations. This process is stopped when the negative log-likelihood estimated with 3-fold cross-validation starts to increase.[2] This is detected by fitting a line through the last m observed costs and stopping when the slope of this line becomes positive for the first time. The number of clusters is found by finding the mixture number for which the cost is the smallest.[3]

[2]We used 3-fold cross-validation as a compromise between speed and accuracy.

[3]We have experimented with a number of schemes here. Initially, we used an exhaustive search (up to an upper bound on the number of clusters), but this was computationally very demanding and analysing the sequence of costs in a number of experiments we observed that costs starts to level off at approximately the correct number of clusters and hence arrived at the above method. The detection of this point is made difficult by both the randomness of the training procedure, the sequences and the limitations of the training procedure. We set m, the smoothing parameter, to 4 experimentally as it seemed to perform the best. We also considered smoothing the costs before searching for their minimum, but that did not change the results significantly. Hence we kept the simpler method.

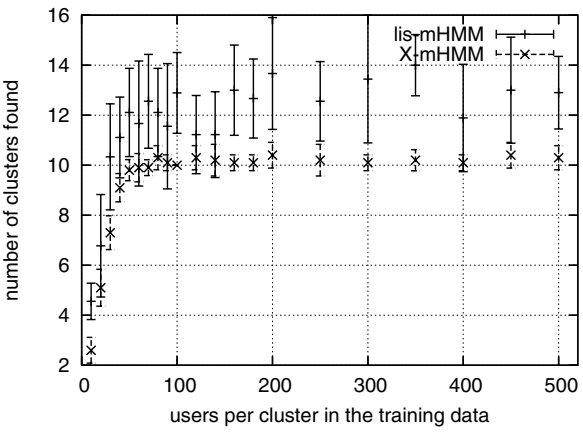

Figure 3. Number of clusters found as a function of the number of training samples.

4.1 Data Generation

The following procedure was adopted as the data generation model: given K, the number of mixtures, the prior p was chosen to be uniform. The HMMs were generated randomly in a way that ensured that they were sufficiently different. This was implemented as follows: let the number of states be fixed at n, plus an exit state. For each state i we decided randomly the number of 'significant' transitions from i (self-transitions are allowed). This number could be 1 or 2. Once this number was decided, transition probabilities were sampled in a manner to ensure that the number of significant transitions from i (having much larger values than the rest of transitions) matches the sampled number. The initial state distribution was generated in a similar way. After this, one distinctive state, called the 'exit' state was added to each model. Next, exit transitions were added to each state in a random manner. The exit probabilities p_e were sampled from the uniform distribution on $[0.05, 0.3]$ and the non-exit transitions were normalized by multiplying them with $1 - p_e$. The same observation matrix was used for each model component of the mixture. The probability observing symbol i in state i was fixed at 0.85 and the observation matrix was chosen to be a circular matrix where off-diagonal elements with $|i - j| > 3$ were set to 0.

In order to closely simulate web-log data where user identity can be used in the clustering process, for each HMM a number of 'users' were generated. Then, the sequences of a given user were generated using the HMM corresponding to the user. The average sequence length

is 7.4. Since in our web-log, users typically have 2-3 sequences, we used a random number of observation sequences where the number of observation sequences was generated by adding one to a Γ distributed random number, where the parameter of the Γ distribution was set to 3. The average number of sequences per user was approximately 5. The identity of users was made available to both algorithms. These used them in semi-VT to assign the sequences of any user to the very same model.

4.1.1 Results on Synthetic Data

Results of the experiments are shown in Figures 3–9. Initially, we were interested in the dependency on the number of training samples for a fixed number of clusters, which in this case was set to 10. Since the generation model is two-level, we give the number of training samples as the number of 'users'. In Figure 3 the number of clusters found is shown as a function of the number of users for both algorithms. The number of states in the HMMs is fixed at 11 (corresponding to the true number of states in the generating models).

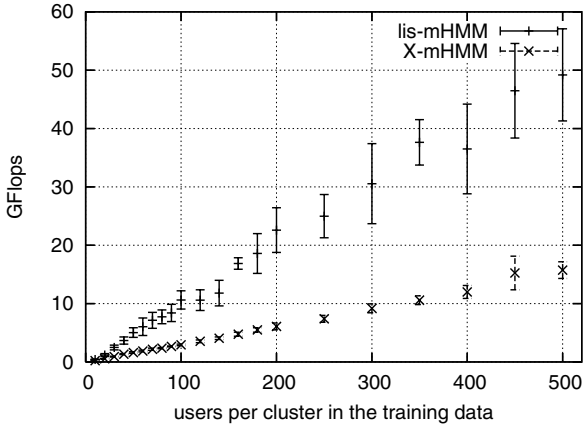

Figure 4. Training cost as a function of the number of training samples.

From this figure we find that after a short transient period, X-mHMM successfully identifies that the number of clusters is 10 (though a closer inspection reveals a slight positive bias). On the other hand, in most of the cases lis-mHMM fails to find the proper number of clusters. Our explanation is that due to the incremental nature of X-mHMM, it is less sensitive to its initialization than lis-mHMM and will tend to have a better chance to identify the 'true' density. In other words, lis-mHMM, due to its "monolithic approach" will often get stuck in local minima making it hard to find the appropriate number of clusters and to identify the true density.[4] We have also checked the clustering of the users in a number of selected cases. It was found that the clustering results of the mixture models returned by X-mHMM largely agreed with the original classification of the users (e.g. when the number of clusters was higher than 10, then typically additional weakly populated clusters were added), and the clustering agreed very well (above 90% correct classification) with the original one. Figure 4 shows the training cost in giga-flops as a function of the number of users sampled per cluster. It can be readily observed that X-mHMM is not only more accurate than lis-mHMM, but it also runs at least 3 times faster than lis-mHMM. Error bars on these figures represent standard deviations over 10 independent runs.

In the next experiment we tested X-mHMM's robustness to the number of states in the HMMs. Since our algorithm does not attempt to find the proper HMM structure, it is important to test its robustness to this parameter. From Figure 5, that shows the number of clusters found as a function of the number of states of the HMMs, we see that X-mHMM is generally robust to this parameter. In particular, increasing the number of states above 11 (the true number of states) yields most of time the correct number (here, just as in the subsequent experiments the number of users per cluster is fixed at 500). On the other hand, lis-mHMM has

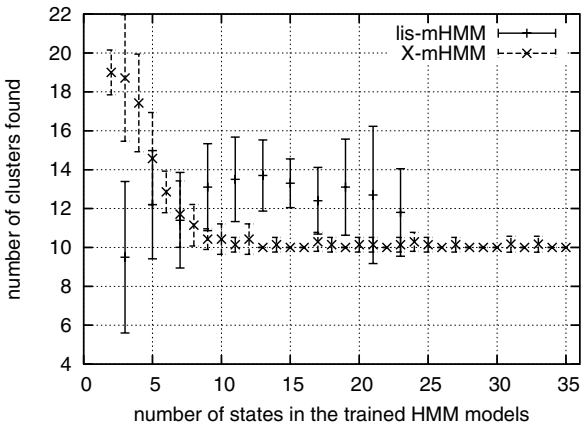

Figure 5. Number of clusters found as a function of the number of states in the HMMs.

difficulties in estimating the proper number of clusters. Results for lis-mHMM are only shown up to the state number 23 due to its large running times. The corresponding graph showing the training-cost of the algorithms is given in Figure 6.[5] Again, X-mHMM is found to have a considerable gain over lis-mHMM. The next figure shows the

[4] Figure 7, to be described soon, confirms this hypothesis.

[5] Theoretically, the training cost of both models should scale quadrati-

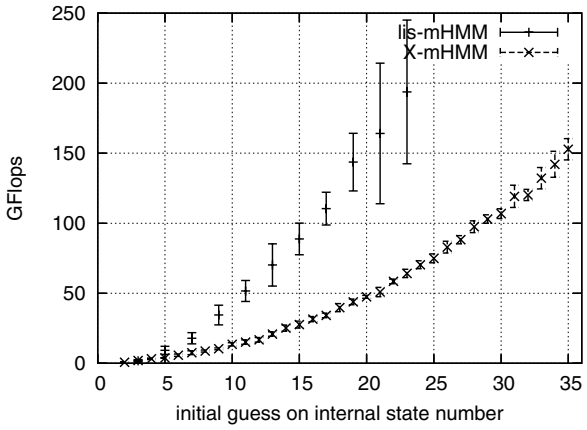

Figure 6. Training cost as a function of the number of states in the HMMs.

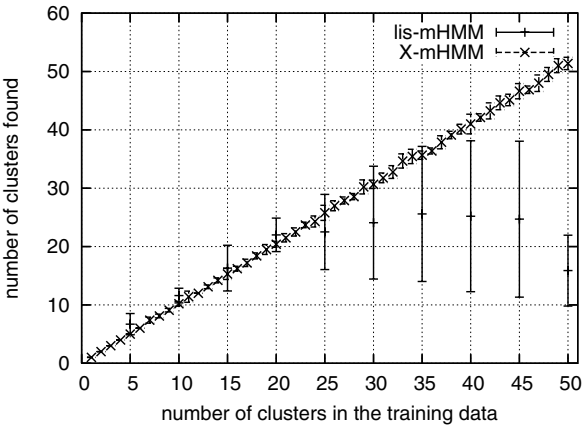

Figure 8. Number of mixtures found as a function of the true number of clusters.

cost of models found (the cost is calculated as the average number of bits per observations corresponding to the optimal compression scheme given the models, i.e., the normed negative log-likelihood of the data where the base of the logarithm is 2).[6] As suggested previously, the models found

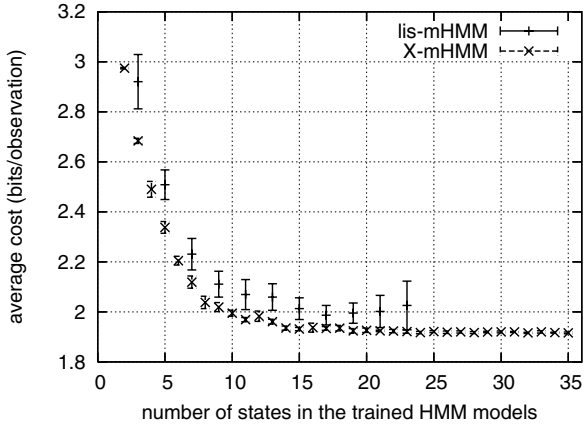

Figure 7. Model cost as a function of the number of states in the HMMs.

by X-mHMM can indeed model the data better than those returned by lis-mHMM. In particular, the costs of the models returned by X-mHMM approach the optimal cost. From these figures we may conclude that X-mHMM is not particularly sensitive to overestimating the number of states needed.

cally with the number of states.

[6]The cost was measured on an independent test set of size equal to that of the training data.

Finally, Figure 8 shows the number of clusters found as a function of the number of clusters used in the generation model, whilst Figure 9 shows the corresponding training costs. The previous conclusions still hold: X-mHMM is not only more accurate than lis-mHMM, but also runs much faster.

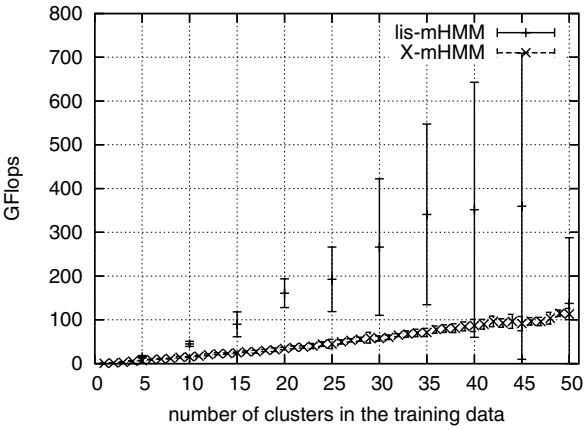

Figure 9. Training cost as a function of the true number of clusters.

4.2 Experiments on Web-log Data

For our experiments we used web-log data of a popular portal. The observations consisted of sequences of addresses of downloaded pages and the identity of users (extracted from cookies) initiating the download. To reduce the observation space, the pages were grouped according to

their 'topic' extracted automatically from their URL. This resulted in an observation space of 150 values. There are 280,000 users in the database and over 800,000 sequences. The total number of observed sequences is ca. 6 million, which amounts to a data set of one month.

Fitting X-mHMM with 10, 20 and 30 internal states (and an additional exit state) showed us, that starting at 20 states the number of clusters stabilized at $5-7$ clusters.

Next, we experimented with the stability of the clusters found. It turned out that cluster boundaries are not as strict as was found in the case of synthetic data. In all investigated cases a mapping between the clusters of mixtures could be identified by looking at simple statistics such as the page visitation frequencies induced by the models. The general finding is that there exist 4 core user clusters, whilst the other clusters are typically smeared over all the other clusters.

The following general labels could be assigned to the most prominent clusters: scientific and political; education and sophisticated news; family; arts and fun.

Given a good estimate of the number of clusters and the number of hidden states, we compared the quality of the model found by X-mHMM with that of a mixture of HMM model with identical number of mixtures and states and trained with semi-VT. We measured the costs (in bits/observation) of the models on a test set comprising of 10% of the whole data, not included in the training set. A cost of 2.45 bits/observation (stddev=0.02 based on 16 runs) was measured for the mixture of HMM model, whilst a cost of 2.33 bits/observation (stddev=0.007 based on 13 runs) was measured for X-mHMM. [7] In this example the run-time of X-mHMM is ca. 9 times longer than that of training the single mixture of HMM for which the "correct" number of states and mixtures is given, whilst X-mHMM is capable of estimating the number of mixtures, and yields better models.

5 Conclusion

In this paper we have proposed X-mHMM, an algorithm for the simultaneous estimation of the number of mixtures and the training of mixture of HMMs. The algorithm is inspired by [7], where a similar algorithm was proposed for clustering vector-space data. For our problems many modifications were necessary to the original algorithm to make it competitive with alternative approaches. In a series of experiments with synthetic and real datasets X-mHMM was found to give excellent performance and run faster than simpler alternatives. In particular, in a large number of cases, it was found that X-mHMM is not only capable of estimating the correct number of clusters with small errors, but it also seems to be successful at avoiding initialization problems of HMMs.

Future work should include the extension of X-mHMM to other probabilistic mixture models, a more thorough study of the performance characteristics of the algorithm (e.g. identifying critical points of the algorithm). According to some authors (e.g. [5]) it is not possible to get reliable estimates of the number of clusters without estimating the correct number of states of the underlying HMMs. In our experiments we have found that at least for our particular data sets, X-mHMM is capable of tolerating an overestimation of the number of states. However, we still think that it would be important to incorporate an algorithm to estimate the number of states as it could lead to even better models.

[7] Using zero-order compression (like Shannon or Huffman codes), the cost is 3.6 bits/observation.

References

[1] L. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8, 1969.

[2] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series. Prentice Hall, 1988.

[3] B. Juang and L. Rabiner. The segmential k-means training procedure for estimating parameters of hidden Markov models. *IEEE Trans. Acoustics, Speech, Signal Processing*, 38:1639–1641, 1990.

[4] L. Kaufman and P. Rousseeuw. *Finding Groups in Data*. John Wiley and Sons, Inc., 1990.

[5] C. Li and G. Biswas. A Bayesian approach to temporal data clustering using hidden Markov models. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 543–550, 2000.

[6] M. F. M. Bicego, V. Murino. Similarity-based classification of sequences using Hidden Markov Models. *Pattern Recognition*, 2004. to appear.

[7] D. Pelleg and A. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, San Francisco, 2000. Morgan Kaufmann.

[8] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, 1989.

[9] P. Smyth. Clustering sequences with hidden Markov models. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, page 648. The MIT Press, 1997.

[10] A. Ypma and T. Heskes. Categorization of web pages and user clustering with mixtures of Hidden Markov Models. In *Proceedings of the International Workshop on Web Knowledge Discovery and Data Mining*, 2002.

[11] S. Zhong and J. Ghosh. A unified framework for model-based clustering. *Journal of Machine Learning Research*, 4:1001–1037, 2003.