STATISTICAL INFERENCE OF TIME-DEPENDENT DATA

A Thesis

Submitted to the Faculty

of

Purdue University

by

Suhas Gundimeda

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF THESIS APPROVAL

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

SYMBOLS

Z	State random variable
z	State variable realization
V_i	A scalar time series
V	Set of time series processes
X_i	A particular sequence of random variables
X	Set of random variables
m	The number of (hidden) states
n	The number of time-series
T	The time horizon

ABBREVIATIONS

TD-tree/tdTree	Time-dependent tree distribution
dist.	distribution
loglik.	Log likelihood
i.i.d	Independent and identically distributed
TS	time series
WLOG	Without loss of generality

NOMENCLATURE

Random variables are capitalized, while realizations are lower case.

Sets are also uppercase letters, meaning should be clear form the context.

P(.) is used as the generic notation for probability distribution,
without referring to any specific parametrization of said distribution.

ABSTRACT

Gundimeda, Suhas M.S., Purdue University, May 2020. Statistical inference of time-dependent data. Major Professor: Christopher J. Quinn.

Probabilistic graphical modeling is a framework which can be used to succinctly represent multivariate probability distributions of time series in terms of each time series's dependence on others. In general, it is computationally prohibitive to statistically infer an arbitrary model from data. However, if we constrain the model to have a tree topology, the corresponding learning algorithms become tractable. The expressive power of tree-structured distributions are low, since only $n-1$ dependencies are explicitly encoded for an n node tree. One way to improve the expressive power of tree models is to combine many of them in a mixture model. This work presents and uses simulations to validate extensions of the standard mixtures of trees model for i.i.d data to the setting of time series data. We also consider the setting where the tree mixture itself forms a hidden Markov chain, which could be better suited for approximating time-varying seasonal data in the real world. Both of these are evaluated on artificial data sets.

1. INTRODUCTION

Many real-world complex systems can be modeled as inter-dependent random processes, such as the stock market, weather, online social networks, and the human brain. Imagine the time series of an industry leader's stock price influencing other stocks in the sector. Loss in confidence in a blue chip index might influence even a strong individual performer's shares. For the weather, time-series of wind, precipitation, and temperature have spatial correlations and some phenomena, such as hurricanes, propagate spatially in time, leading to dependencies in such time series. The brain's functional state during an activity could be the result of a particular set of brain regions activating together. The brain while reading a philosophical book might be a juxtaposition of a comprehension network and a reflective, diffuse mode network. Reading a fantasy book might correspond to a cross between comprehension and visualization networks. For many of such complex systems, scientists desire to identify and model inter-dependencies between the time-series. We can observe some of these processes, using stock exchange application programming interfaces (APIs), recording rain gauge measurements and measuring fMRI data while performing different activities, respectively. We might now ask several useful questions about these scenarios, we can try to build a better hedging algorithm for investors. We could predict rainy days and areas to aid in crop planning. We could try and understand the factors determining the networks of the brain across effective and ineffective habits.

Probabilistic graphical models are a classical answer to representing and extracting meaning from such data. They encode statistical relationships among a given set of variables or time-series using a graph structure.

Even with data, identifying which probabilistic graphical model best fits the data is challenging. The number of topologies the probabilistic graphical model could correspond to are exponential in the number of time-series. Learning the exact model that explains these systems thus might not be computationally tractable, or we might not have the quantity of data that would enable us to do it. We may then seek to approximate these datasets with models that compress the data a lot, but are still expressive enough to explain and predict.

There are two major lines of research of how to restrict the class of probabilistic graphical models to make structure learning computationally tractable. The first is to restrict the families of distributions being considered. For instance, multivariate linear auto-regressive time-series are comparatively easy to work with and there are well-developed, computationally efficient regularization techniques analogous to the LASSO for i.i.d. data, such as [1–5]. While some of these methods apply to non-linear autoregressive time-series, they nonetheless make assumptions about the parametric family or about the spectral properties of the distributions.

The second line of research involves making assumptions about the topology of the graphical models. For instance, learning a probabilistic graphical model for time-series where the dependencies form a *tree structure* is a known tractable task [6]. But tree distributions are not deemed expressive enough for the arbitrary strands of influence in the real world, because of the restricted number of dependencies.

We propose to use an ensemble of models, where each component model is easy to learn - for example, a tree structure of dependencies - and having the whole model be a convex combination of these trees, a *mixture of trees*, potentially providing a large amount of expressive power. For i.i.d data, it is known that mixtures of trees are both efficient to learn and infer from, while offering good expressive power [7].

1.0.1 Our Contribution

Our current work describes *mixtures of time-dependent trees*, for application on time series data. We provide a learning algorithm for mixtures of time-dependent trees, extending work on mixtures of trees for i.i.d data [7]. We attempt to characterize empirical performance on artificial datasets. Finally, we extend the framework to hidden Markov models with time-dependent tree observables, and characterize its performance. Unlike [8], which extends mixtures of trees beyond i.i.d. data with a tree on the underlying dynamic Bayesian network, our proposal does not penalize self-dependence and does not allow for unrealistic instantaneous dependencies.

1.0.2 Section Overview

In Chapter 2 we describe the model and algorithms for approximating observed multivariate data as mixtures of time-dependent trees. Also, the primary notation and formulation is introduced. Chapter 3 deals with the case where the distribution of the data itself changes over time, modeled as a Markov process, drawing on the hidden Markov model (HMM) literature. This extends the mixture of time-dependent trees model to HMMs where the emission distribution is a time-dependent tree. The evaluation of these models is presented towards the end of each chapter.

1.1 Related work

1.1.1 Graphical model selection

Bayesian networks or Markov random fields are arguably the two most popular probabilistic graphical models for i.i.d. random variables, as described in Chapter 8 of [9]. Learning of the best graphical models given data is a well studied topic, and algorithms exist for many families of parametric distributions, such as Gaussian and Ising models [10], as well as for non-parametric distributions. See Chapter 26 of [11] for more detail.

We are specifically looking for approximations whose topology is sparse, like tree structures. For learning tree distributions, that is graphical models whose topology is a spanning tree, seminal work by Chow and Liu showed that the maximum-likelihood tree approximation (second order product distribution) for i.i.d data is given by the maximum weight spanning tree (MSWT) algorithm, with mutual information as weights [12,13]. This was more recently extended to time-dependent data, for minimum-cost arborescences and directed mutual information as weights in [6,14], which is crucial to this work.

There are also recent works on high dimensional loopy model selection, which can be classified into convex optimization based approaches and non convex local approaches ([15], [16], [17], [18]). There is also variable selection using the Lasso [19]. None are directly applicable to learning mixtures of graphical models, however.

A new approach is based on *rank tests*, a generalization of conditional independence tests, for estimating the graph structure [20]. This approach is efficient with sparse graphs, such as tree and bounded degree approximations. This can also be used for learning mixtures of trees.

1.1.2 Learning mixtures of trees

Meila and Jordan proposed a procedure to learn mixtures of spanning trees for i.i.d. data, based on the Expectation-Maximization algorithm [7]. Our work primarily builds on this, extending it to time-dependent data, and then to non-i.i.d Markovian hidden states. [8] proposed mixtures of trees for dynamic Bayesian networks, although as we later discuss, this prohibits self-dependence for all the time-series (without requiring them to be independent) and also results in instantaneous dependencies. Our work looks for trees not on the dynamic Bayesian network, but instead on a network where each node is a time-series. [21] proposes a Gaussian mixture of trees to model multi-modal time series data. Anandkumar et al (2012) propose an alternative to the classical EM approach to learning mixtures of trees distributions, and seminally, one that has provable convergence guarantees [20]. Roughly, their procedure involves a generalization of conditional independence tests to efficiently estimate a *union (of mixture components) graph* structure. This is used to obtain a series of mixtures of product distributions. Spectral decompositions are used to obtain the corresponding pairwise marginals, upon which Chow-Liu [12] is applied to get the final tree approximations. Some restrictions of a minimum number of samples and a full rank probability matrix exist.

1.1.3 Markovian hidden state models

The aforementioned mixture model can be viewed as randomly picking a component distribution to draw samples from using the mixture coefficients as probabilities. From this perspective, which component distribution is selected is an i.i.d. process.

Hidden Markov models have an unobserved Markovian state process evolving with time, which influences the distribution of observed variables. For mixture models, the component distribution selection can be extended from i.i.d. process to a (hidden) Markov process.

A pedagogical review of the traditional formulation and techniques in hidden Markov models is provided in chapter 13.2 of [9]. Our derivation in Section 3.3 runs parallel to this. Rabiner (1989) provides a tutorial on HMMs of various types [22]. [23] provides a treatment of statistical inference from data of hidden Markov models with autoregressive (linear dependence across time) observed time series using the Expectation-Maximization algorithm, in the context of speech recognition. In our work, the emission distribution instead has the form of time-dependent trees.

Another interesting approach [8] to model time series processes is using Hidden Markov Models (HMMs) with Chow-Liu structures as emissions. The state variable is used for modeling temporal structure of data and Chow-Liu trees for simultaneous interdependencies (HMM-CL). The work goes on to describe conditional Chow Liu (HMM-CCL) structures to model dependencies across time (and simultaneous dependencies) in time series processes. This is related to Chapter 3 of this work. One important difference is that HMM-CCL models include instantaneous correlations and do not naturally account for each time series self-dependence.

There are several recent works on mixtures of Markov chains, each of which propose specific constraints to learn the best approximation. One approach used mixtures of low dimensional Markov chains to avoid the curse of dimensionality, and proposed a Bayesian approximate estimator [24]. Another dealt with the problem of reconstructing Markov chain mixtures from given observation trails, proposing a spectral method which outperforms EM and is faster in some regimes [25]. Yet another work on learning mixtures of Markov chains from data proposed structural constraints to make the problem tractable, and a constrained least-squared method to learn the model [26].

2. MIXTURES OF TREES FOR TIME-SERIES DATA

2.1 Introduction

Our goal is to develop sparse models of interdependencies between time series and to develop algorithms to learn those models from observational data. Specifically, we present mixtures of time-dependent trees models for self-dependent and interdependent time series data. Instead of learning a single, sparse topology, which may yield poor models, we will learn mixtures of sparse topologies, specifically mixtures of directed, rooted, trees. We next describe the model class in terms of joint distributions. For clarity we first review the i.i.d. random variable case, then expand to the time-series process case.

2.2 Background: product, tree and mixture distributions for i.i.d data

Let $X = (X_1, X_2, X_3, \ldots X_n)$ denote n random variables with a joint distribution $P(X)$. We can re-write the distribution as a product of conditional distributions using the chain rule,

$$P(X) = \prod_{i=1}^{n} P(X_i | X_1, X_2, \ldots X_{i-1}).$$

Further, X_i may not have a strong dependence on all of the variables $X_1, X_2, \ldots X_{i-1}$. We can consider approximating $P(X)$ using only up to second-order dependencies, such as

$$\widehat{P}(X) = P(X_1)P(X_2 \big| X_1)P(X_3 \big| X_1)P(X_4 \big| X_2, X_1)\ldots$$

We can visualize these dependencies between variables graphically with nodes depicting variables and edges depicting dependencies, as shown in Figure 2.1. Let $G(X, E)$ denote this graph with vertex set X and edge set E,

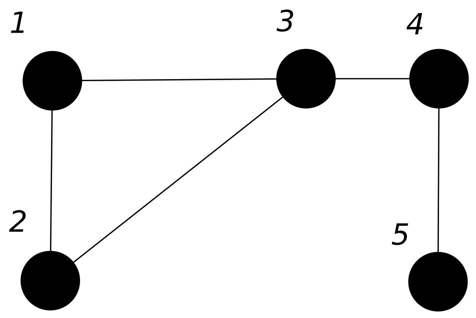


Figure 2.1. An graphical model of an example i.i.d product distribution. A connection implies conditional dependence, as in a "Markov random field". This graph conveys that the distribution could be decomposed as $P(X_5)P(X_4\big|X_5)P(X_3\big|X_4)P(X_1\big|X_3)P(X_2|X_1, X_3)$

.

Now, consider the case where for a certain variable ordering, every variable is (approximated to be) conditionally dependent on a maximum of one other variable and the graphical model is connected (e.g. the graph is a spanning tree). For a variable X_i, let $pa(X_i) \in \{X_{i-1}X_{i-2}\dots X_1\}$ denote the variable that X_i is conditionally dependent on. Let $T(X)$ denote the tree distribution for a distribution $P(X)$.

$$P(X) = \prod_{i=1}^{n} P(X_i|X_{i-1}X_{i-2}\dots X_1)$$

$$T(X) = P(X_1)\prod_{i=2}^{n} P(X_i|pa(X_i))$$

We assume without loss of generality[1] that X_1 is the root node, and $pa(X_i)$ denotes the parents of X_i. Succinctly, we call such distributions "tree distributions". Given observational data, learning the full distribution may be prohibitive for computational and statistical reasons. Instead, we may seek to learn a simple, sparse model such as a tree distribution. The goal is then to learn the best possible tree distribution (approximation), in the maximum-likelihood sense.

Chow and Liu [12] proposed an algorithm to identify the best tree distribution. We first estimate the *mutual information* (MI) between each pair of variables. For variables X_i and X_j, the mutual information is the expected log-likelihood ratio between the pairwise distribution $P(X_i, X_j)$ and the product of the marginals,

$$I(X_i; X_j) = \mathrm{E}_{P(X_i,X_j)}\left[\log \frac{P(X_i, X_j)}{P(X_i)P(X_j)}\right].$$

The mutual information can be thought of as a non-parametric generalization of linear correlation. It is zero iff X_i and X_j are (marginally) independent. After computing the mutual information for each pair of variables, we next construct a complete graph among all variables with edge weights being the MIs. Chow and Liu [12] showed that for i.i.d data, the maximum weight spanning tree that can be obtained from such a complete graph, maximizes the likelihood over all possible tree distributions.

Tree distributions are relatively easy to learn (compared to learning the full distribution) since they only require pairwise distributions. However, that also limits

[1]We can relabel the variables with the root node to be node 1.

their expressiveness. To better approximate data, Meila et al [7] proposed using mixtures of these trees, leading to a convex combination of several tree distributions. Let $T_k(X)$ denote the kth tree distribution in the mixture and let λ_k denote its mixture coefficient. Let m denote the number of tree distributions in the mixture. Then the mixture distribution, denoted as $Q(X)$, can be expressed as

$$Q(X) = \sum_{k=1}^{m} \lambda_k T_k(X). \tag{2.1}$$

We can also express the mixture as having a hidden state random variable Z that can take m values (one for each tree distirbution), with $\lambda_k = P(Z = k)$.

Meila et al [7] proposed an Expectation-Maximization (EM) algorithm to learn the best mixture of trees. It is analogous to EM algorithms for learning other mixtures, like mixtures of jointly Gaussian distributions. We will now move onto an analogous description of models for time-series data, in which we will address each step in more detail.

2.3 Background: Product and tree distributions for time series data

We will propose an algorithm to learn mixture of time-dependent tree models. This uses the algorithm for learning of a single time-dependent tree model as as a subroutine. This section describes this background.

2.3.1 Product distributions for time series data

Consider n time-series, i.e. random processes $V = \{V_1, V_2, \ldots V_n\}$ which are described by the joint distribution $P(V)$.

We can apply the chain rule both over time and over space. First, we factorize over time

$$P(V) = \prod_{t=1}^{T} P(V(t)|V(1), \ldots V(t-1))$$

In many real-world systems, when there are no confounding factors, the future may only depend on the recent past.

Assumption 1 *The joint distribution $P(V)$ is a first-order Markov chain,*

$$P(V) = \prod_{t=1}^{T} P(V(t)|V(t-1)).$$

Remark 1 *All of our work can be easily extended to higher Markov orders, though the computational and possibly sample complexities will increase.*

We can next apply the chain rule over space for each time t,

$$P(V) = \prod_{t=1}^{T} \prod_{i=1}^{n} P(V_i(t)|V(t-1), V_1(t), \ldots, V_{i-1}(t)).$$

Notice that the conditioning is on the whole system's past $V(t-1)$ and other processes at time t. We further assume that with a high-enough sampling rate and no confounding factors, the time series at time t are conditionally independent given the previous system state $V(t-1)$.

Assumption 2 *The joint distribution $P(V)$ factorizes as*

$$P(V) = \prod_{t=1}^{T} \prod_{i=1}^{n} P(V_i(t)|V(t-1)).$$

In general, each of the terms in the product may further simplify. For instance, if process i only depends on it's own past and the past of process j, that means that for all time t,

$$P(V_i(t)|V(t-1)) = P(V_i(t)|V_i(t-1), V_j(t-1)).$$

A generic case is illustrated in Figure 2.2. Also, a "time-unraveled" visualization with explicit strictly causal time dependence is illustrated in Figure 2.3. We omit self-loops (for representing self dependence over time) in the compact representation.

We will explore approximations where we only allow each time-series to depend on its own past and the past of at most one other time-series.

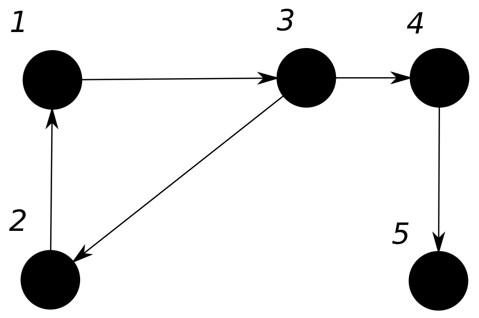


Figure 2.2. An illustration of a time dependent product distribution. Each arrow represents a dependence. This graph conveys that the distribution is $P(V(t)|V(t-1) = P(V_1(t)\big|V_1(t-1)V_2(t-1)) \times P(V_2(t)\big|V_2(t)V_3(t-1)) \times P(V_3(t)\big|V_3(t)V_1(t-1)) \times P(V_4(t)\big|V_4(t-1)V_3(t-1)) \times P(V_5(t)\big|V_5(t-1)V_4(t-1))$. The directed cycle indicates feedback.

2.3.2 Time dependent tree distributions

We now look at product distributions over time series where the interdependencies form a tree topology. Specifically, the topology is a directed, rooted tree where all edges point away from the root (also known as an *arborescence* or a *branching*).

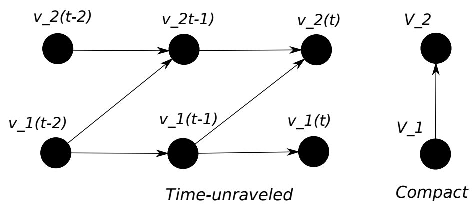


Figure 2.3. Two representations of the same TD-tree, with the left figure being unraveled over time, and the right one being the compact representation. Each arrow represents a dependence. Self-dependency always exists. Equivalently, $V_1 \rightarrow V_2$, which implies the distribution $P(V_2) = P(V_2(t) \big| V_2(t-1), V_1(t-1))$ for time series V_1, V_2. Also, $V_1,\ V_2$ form a Markov chain.

Let $pa(V_i)$ denote the parent process of V_i in the topology. A *time-dependent tree distribution* T is then defined as the product distribution

$$T(V) := \prod_{t=1}^{T} \prod_{i=1}^{n} P(V_i(t)|V_i(t-1), pa(V_i)(t-1)). \tag{2.2}$$

An example of the product distribution and the corresponding arborescence is illustrated in different forms in Figure 2.3. Two more examples of arborescences in compact form are illustrated in Figure 2.4.

We next discuss how such a model can be learned from data. The expressiveness of this model is described in Section 2.6.

2.3.3 Learning of time-dependent tree distributions

We are given a dataset of n random time series processes $V = \{V_1, V_2, \ldots V_n\}$ across T time steps $V_i = V_i(1), V_i(2), \ldots V_i(T)$, a $V_{n\times T}$ realization matrix. Let $V(t)$

denote the set of all processes at time t . The goal of this section is to describe an algorithm that learns the rooted directed tree with all edges directed away from root (an arborescence) topology and corresponding distribution that best approximates the data. Let $\mathcal{T}$ denote the set of all distributions over V. We quantify "best" as the tree distribution with the maximum likelihood (ML) of generating the data, which is equivalent to the maximum log likelihood model because of the monotonicity of log function. This ideal tree distribution T^* can be written as:

$$T^* = \underset{T \in \mathcal{T}}{\arg\max} \ \log \ T(V) \tag{2.3}$$

An algorithm to solve Equation 2.3 was proposed by Quinn et al [6] and is summarized in Algorithm 1. We will later use it as a subroutine. It is analogous to algorithm Chow-Liu for i.i.d data. For time-dependent data, with directed information (a time-series analog of mutual information) as weights, the maximum weight spanning arborescence (also called *optimum branching*) maximizes the likelihood among all possible directed rooted trees [6].

Consider a complete graph $G_c(V, E_c)$ that has directed edges between all possible pairs of vertices, each with a real weight equal to the the directed information along that edge. Learning the optimal spanning arborescence given real weights for edges is described by Edmonds-Chu (or Edmonds') algorithm [14, 27], which fills the role that MWST does in the i.i.d data case.

The runtime of the standard Edmonds' algorithm is $\mathcal{O}(VE)$. This has been optimized, bringing down runtime to $\mathcal{O}(E \log V)$ for sparse and $\mathcal{O}(V^2)$ for dense graphs [28] with a later work producing a $\mathcal{O}(E + \log V)$ algorithm [29].

To compute the directed information, we first compute the empirical distributions $P(V_i \big| pa(V_i)), P(pa(V_i))$. For implementation ease, we assume discrete time series taking values from Ω_V. We decompose the required conditional empirical estimators into more flexible joint distributions:

$$P(V_i(t) \big| V_j(t-1)V_i(t-1)) = \frac{P(V_i(t), V_i(t-1), V_j(t-1))}{P(V_i(t-1), V_j(t-1))} \qquad \text{Where } V_j \text{ is a parent of } V_i$$

$$P(V_i(t) \big| V_i(t-1)) = \frac{P(V_i(t), V_i(t-1))}{P(V_i(t-1))} \qquad \text{Where } V_i \text{ is a root node.}$$

Here we consider Markov order 1 with self-dependence and one parent. Massey [30] defined directed information for communication channels with synchronized channel outputs and inputs:

$$I(V_j \to V_i) := \sum_{t=1}^{T} I(V_j; V_i(t)|V_i(t-1)) \qquad \text{For time series } V_j \to V_i$$

We modify this to compute strictly causal directed information between each pair of time series, with Markov order 1:

$$\begin{aligned} I(V_j \to V_i) &= I(V_j(t-1); V_i(t)|V_i(t-1)) \\ &= \sum_{(a,b,c)\in\Omega_V^3} P(V_i(t)=a, V_i(t-1)=b, V_j(t-1)=c) \\ &\times \log \frac{P(V_i(t)=a, V_i(t-1)=b, V_j(t-1)=c)P(V_i(t-1)=b)}{P(V_j(t-1)=c, V_i(t-1)=b)P(V_i(t)=a, V_i(t-1)=b)} \end{aligned}$$

Here $V_i(t), V_i(t-1), V_j(t-1)$ take on all the possible values in their sample space. Thus this step scales as $\mathcal{O}(\Omega_V^3)$. We could also iterate over just the realizations present in data, as the rest would contribute 0 to the sum. Then, this step scales as $\mathcal{O}(T)$. The same measure exists for continuous distributions of processes, with a description in [6]. We can now compute the best topology of the graph (optimal arborescence).

The full algorithm to learn time dependent tree distributions from data is Algorithm 1. Given a dataset of n discrete time series processes over T time steps, we hope to return a model where each node's stationary distribution across time specifies self-dependence as well as dependence on one other node (except for the root) i.e. a tree product distribution for time series that is the best structure and parameters in the maximum likelihood sense.

Computational complexity

Computing all of empirical marginal distributions required for directed information takes $\mathcal{O}(n^2T)$ steps total. The total number of steps to compute all the directed information calculations is $\mathcal{O}(n^2|\Omega|^3)$. Finding the optimal spanning arborescence

Algorithm 1: ChowLiuTS: Algorithm for learning a time dependent tree distribution from time series data

```
Input  : Dataset $V$ of $n$ variables varying across $T$ time steps.
         Edmonds' algorithm
Result: Graph structure and distribution corresponding to the best directed
        graph that describes the data
1 Compute/Estimate distributions $P(V_i(t), V_i(t-1), V_j(t-1)$ for all
    $i, j \in \{1, \ldots, n\}$ using any estimator, for example, empirical.
2 Compute directed information between each ordered pair of nodes from given
    distributions of data.
3 G ← Edmonds(complete graph with directed information edge weights)
4 Return G and corresponding subset of the distributions
    $P(V_i(t) \big| V_i(t-1), pa(V_i)(t-1))$
```

(Edmonds' algorithm), for the best implementation, is $\mathcal{O}(n^2 + \log n)$. Thus learning of time dependent tree distributions comes out to be $\mathcal{O}(n^2(T + \left|\Omega\right|^3))$. Another method of computing directed information is using asymptotic equipartition theorem (AEP; see Ch. 3 of [31]), iterating over the realized samples (observations) rather than the entire sample space. With such an implementation, the computation scales linearly with T, bringing the total to $\mathcal{O}(n^2 T)$.

2.4 Mixtures of time-dependent trees

Consider the example of rainfall prediction from the introduction, where we want to predict rainfall at multiple locations. We can imagine that multiple consistent wind patterns, such as the polar and pacific jetstreams, El Nino, and others each have an effect on how weather is affected across the north American landmass. Tree distributions alone would be insufficient to model such a system. We now bring in a hidden state variable that "mixes" multiple such tree distributions together, so that we may describe the result of different weather systems and patterns of rainfall using a convex combination of many time-dependent tree distributions.

Mixtures of time-dependent trees are potentially much more expressive than tree models, as they have many more parameters available to approximate a distribution.

We define mixture models for time-series analogous to mixtures in the i.i.d. variable setting (2.1). Like the i.i.d. setting, we can view the mixture distribution as characterized by a latent, i.i.d. process $Z(t)$, that chooses between the m "mixture components" at each time step. Then the mixture coefficients $\{\lambda_k\}_{k=1}^m$ correspond to the probabilities

$$P(Z(t) = k) = \lambda_k.$$

See Figure 2.4 for a diagram of the graphical model, with each time step having dependencies based on the corresponding tree distribution $T_k(Z)$. While this view of mixture models (as time-varying sparse conditional distributions) is not necessary, it is helpful for explaining the model.

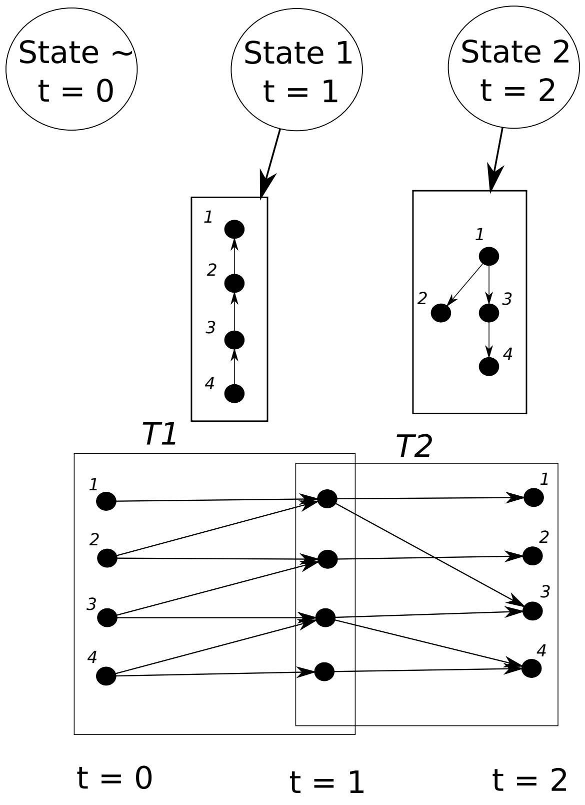


Figure 2.4. An illustration of how i.i.d hidden states affect the dependencies of variables. Note that even if the graph edges remain the same, but the distribution describing the dependence between two nodes (say) changes, we must still consider the changed entity as a new, different tree distribution.

Thus the product distribution of our model can be written as

$$\begin{aligned} Q(V,Z) &= Q(Z)Q(V|Z) \\ &= \prod_{t=1}^{T} \lambda_{Z(t)} T_{Z(t)}(V(t), V(t-1)) \\ &= \prod_{t=1}^{T} \lambda_{Z(t)} \prod_{i=1}^{n} P(V_i(t)|V_i(t-1), pa^{T_{Z(t)}}(V_i)(t-1)), \end{aligned} \tag{2.4}$$

where $T_{Z(t)}$ is a time dependent tree distribution from subsection 2.3.2 and $pa^{T_{Z(t)}}(V_i)$ denotes time series i's parent in the tree distribution $T_{Z(t)}$.

We can marginalize over the i.i.d. sequence Z to get

$$\begin{aligned} Q(V) &= \sum_{z\in\{1,\ldots,m\}^T} Q(z,V) \\ &= \sum_{z\in\{1,\ldots,m\}^T} \prod_{t=1}^{T} \lambda_{z(t)} T_{z(t)}(V(t), V_i(t-1)), \quad (2.5) \\ &= \prod_{t=1}^{T} \sum_{k=1}^{m} \lambda_k T_k(V(t), V_i(t-1)) \\ &= \prod_{t=1}^{T} \sum_{k=1}^{m} \lambda_k \prod_{i=1}^{n} P(V_i(t)|V_i(t-1), pa^{T_k}(V_i)(t-1)). \quad (2.6) \end{aligned}$$

2.5 Learning of mixtures of time-dependent trees

Learning the best-fitting mixture of trees is more challenging than learning a single tree distribution model. Viewing the mixture model as having a hidden variable Z, to learn the model we need to estimate the the best tree distributions for each of subset of data that were drawn from the same component while estimating the best partitioning of the data into such subsets. The classical method for solving this for i.i.d data, as proposed by Meila and Jordan [7], used the expectation-maximization (EM) algorithm. This is an iterative algorithm, which at each step improves upon the expected log likelihood of the current model, with respect to random data, and converges to a local maxima of log likelihood [32]. We propose a similar strategy for learning of mixtures of time-dependent trees from time-series data.

We are given a dataset of n discrete time processes $V = \{V_1, V_2 \ldots V_n\}$ across T time steps $\{V(1), V(2) \ldots V(T)\}$, where $V(t)$ represents the vector of values of all time series at time t. We propose an algorithm that learns the mixtures of trees model that best approximates the data V, in terms of maximum likelihood. This best model can be mathematically expressed as:

$$\begin{aligned} Q^*(V) &= \underset{\{\lambda_k, T_k\}_{k=1}^m}{\arg\max} \prod_{t=1}^{T} \sum_{k=1}^{m} \lambda_k T_k(V(t)|V(t-1)) \\ &= \underset{\{\lambda_k, T_k\}_{k=1}^m}{\arg\max} \prod_{t=1}^{T} \sum_{k=1}^{m} \lambda_k \prod_{i=1}^{n} P(V_i(t)|V_i(t-1), pa^{T_k}(V_i)(t-1)). \end{aligned} \tag{2.7}$$

However, this can be intractable to solve directly. Instead, following the standard outline of EM methods, if we assume that the hidden process $Z(t)$ could be observed, and using Equation 2.4 and an indicator function $\mathbb{1}_{Z(t)=k}$ indicating whether the hidden process $Z(t)$ is state k,we can express the *"complete" log likelihood.* Using (2.4)

$$\begin{aligned} l_{complete}(V, Z \big| Q(V, Z)) &:= \log Q(V, Z) \\ &= \log \prod_{t=1}^{T} \lambda_{Z(t)} T_{Z(t)}(V(t), V(t-1)) \\ &= \sum_{t=1}^{T} \log \lambda_{Z(t)} T_{Z(t)}(V(t), V(t-1)) \\ &= \sum_{t=1}^{T} \log \lambda_{Z(t)} + \log T_{Z(t)}(V(t), V(t-1)) \\ &= \sum_{t=1}^{T} \sum_{k=1}^{m} \mathbb{1}_{Z(t)=k}(\log \lambda_k + \log T_k(V(t), V(t-1))). \end{aligned} \tag{2.8}$$

where (2.8) uses the indicator function.

The EM algorithm uses this *complete* log likelihood as a proxy for the incomplete likelihood Equation 2.7, as it is generally easier to maximize w.r.t changing model in the maximization (M) step. The computation of the expected value of this expression given the data and with respect to unobserved data is part of the expectation (E) step. This gives us a measure of fitness, the quantity we want to maximize over the course of the algorithm. In the process of this, we also get the proportion of each mixture component at every time point. These proportions are values with which we can (soft) partition the dataset into multiple datasets each generated by a mixture component, in other words, a "clustering" of data into each mixture component. This is then used in the M-step, where each partition is used to learn each mixture component separately.

Since we observe the time-series V but not the latent i.i.d. state sequence Z, we will work with the expected value of Equation 2.8 under the conditional distribution $Q(Z|V)$,

$$\begin{aligned} & E_{Q(Z|V)}\left[l_{complete}(V, Z \mid Q(V,Z)) \middle| V \right] \\ &= E_{Q(Z|V)}\left[\sum_{t=1}^{T} \sum_{k=1}^{m} \mathbb{1}_{Z(t)=k} (\log \lambda_k + \log T_k(V(t), V(t-1))) \middle| V \right] \qquad (2.9) \\ &= \sum_{t=1}^{T} \sum_{k=1}^{m} E_{Q(Z|V)}\left[\mathbb{1}_{Z(t)=k} \middle| V \right] (\log \lambda_k + \log T_k(V(t), V(t-1))) \qquad (2.10) \end{aligned}$$

where (2.9) follows from linearity of expectation.

Using Bayes' rule, the expectation of the delta function conditioned on data V can be expressed as

$$\begin{aligned}
E_{Q(Z|V)}\left[\mathbb{1}_{Z(t)=k}\middle|V\right] &= Pr(Z(t)=k\big|V) \\
&= Pr\big(Z(t)=k\big|V(t),V(t-1)\big) \\
&= \frac{Pr\big(Z(t)=k,V(t)|V(t-1)\big)}{Pr\big(V(t)|V(t-1)\big)} \\
&= \frac{Pr\big(Z(t)=k\big|V(t-1)\big)\times Pr\big(V(t)\big|Z(t)=k,V(t-1)\big)}{Pr\big(V(t)|V(t-1)\big)} \\
&= \frac{Pr\big(Z(t)=k\big)\times T_k(V(t),V(t-1))}{Pr\big(V(t)|V(t-1)\big)} \\
&= \frac{\lambda_k T_k(V(t),V(t-1))}{\sum_{k'=1}^{m} Pr\big(Z(t)=k',V(t)|V(t-1)\big)} \\
&= \frac{\lambda_k T_k(V(t),V(t-1))}{\sum_{k'=1}^{m}\lambda_{k'}T_{k'}(V(t),V(t-1))}.
\end{aligned}$$

This last quantity is the posterior probability of the hidden state being k given the time-dependent data at time t and $t-1$. We next define several quantities to parameterize the above:

$$\gamma_k(t) := \frac{\lambda_k T_k(V(t),V(t-1))}{\sum_{k'=1}^{m}\lambda_{k'}T_{k'}(V(t),V(t-1))} \tag{2.11}$$

$$\Gamma_k := \sum_{t=1}^{T}\gamma_k(t) \tag{2.12}$$

$$P_k(t) := \frac{\gamma_k(t)}{\Gamma_k} \tag{2.13}$$

$\gamma_k(t)$ is the posterior probability $Pr(Z(t)=k\big|V(t),V(t-1))$ of time step t being from mixture component k; its formula follows from Bayes' rule. The sum Γ_k can be interpreted as the total number of data points generated from tree distribution T_k. $P_k(t)$ is the normalized soft partitioning of data point t, it gives the posterior probability that it was generated by tree distribution k per datapoint generated by mixture component k.

In the E-step of the EM algorithm, we compute $E_{Q(Z|V)}\left[\mathbb{1}_{Z(t)=k}\middle|V\right]=\gamma_k(t)$ using (2.11) with fixed model parameters $\{\lambda_k,T_k\}_{k=1}^{m}$.

Rewriting Equation 2.10 in terms of $\gamma,\ \Gamma,\ P_k(t)$, and using linearity of expectation

$$\begin{aligned}
E(l_{complete}) &= \sum_{t=1}^{T}\sum_{k=1}^{m} E_{Q(Z|V)}\left[\mathbb{1}_{Z(t)=k}\middle|V\right](\log\lambda_k + \log T_k(V(t),V(t-1))) \\
&= \sum_{t=1}^{T}\sum_{k=1}^{m}\gamma_k(t)\,(\log\lambda_k + \log T_k(V(t),V(t-1))) \\
&= \sum_{k=1}^{m}\left(\sum_{t=1}^{T}\gamma_k(t)\right)\log\lambda_k + \sum_{t=1}^{T}\sum_{k=1}^{m}\gamma_k(t)\log T_k(V(t),V(t-1)) \\
&= \sum_{k=1}^{m}\Gamma_k\log\lambda_k + \sum_{t=1}^{T}\sum_{k=1}^{m}\gamma_k(t)\log T_k(V(t),V(t-1)) \\
&= \sum_{k=1}^{m}\Gamma_k\log\lambda_k + \sum_{k=1}^{m}\Gamma_k\sum_{t=1}^{T}P_k(t)\log T_k(V(t),V(t-1)) \qquad (2.14)
\end{aligned}$$

The M-step involves optimizing (2.14) with respect to the parameters $\{\lambda_k, T_k\}_{k=1}^{m}$ for fixed posterior probabilities $E_{Q(Z|V)}\left[\mathbb{1}_{Z(t)=k}\middle|V\right] = \gamma_k(t)$. Following standard analysis for EM, we form the Lagrangian of (2.10) by including a term $-\nu(\sum_{k=1}^{m}\lambda_k - 1)$ for the constraint that the λ_k's must sum to 1, and differentiating with respect to $\lambda_{k'}$ and setting it equal to 0, we obtain

$$\lambda_{k'} = \frac{\Gamma_{k'}}{\nu}.$$

Setting the derivative of the Lagrangian with respect to ν equal to 0, we get back the constraint $\sum_{k=1}^{m}\lambda_k = 1$. These two equations then lead to the following update equation:

$$\lambda_k = \frac{\Gamma_k}{T} \qquad \forall k \in \{1,2,...m\}. \qquad (2.15)$$

The optimal values for λ_k are independent of the distribution models $\{T_k\}_{k=1}^{m}$. Considering the information we have now ($P_k(t)$) regarding the normalized posterior probability of a time point t being drawn from the tree distribution corresponding to state k, we can use this to "weigh" each data point to obtain a new dataset. This is not a hard clustering, as the same sample can contribute to multiple components. For example, with the empirical estimator:

$$T_k\big(V_p(t)=a, V_q(t-1)=b, V_r(t-1)=c\big) = \sum_{t=1}^{T} P_k(t)\mathbb{1}_{V_p(t)=a}\mathbb{1}_{V_q(t-1)=b}\mathbb{1}_{V_r(t-1)=c}$$

Thus, given the pairwise marginals for each time series for each component, we can use the same Chow-Liu-TS Algorithm 1 to separately learn each mixture component. With m runs of the Chow-Liu-TS algorithm, one for each mixture component, we obtain all m topologies and parameters of the mixture components. This completes the M-step, and we now have a different model to go through the E-step with. We repeat this until convergence or some stopping criteria is reached. Here, we use the minimal fractional change in log-likelihood across iterations as the stopping criteria. The effect of changing this threshold is shown in Figure 2.8.

The EM algorithm to learn mixtures of time-dependent trees is summarized in algorithm 2.

Algorithm 2: Learning of mixtures of trees with time series data

```
Data: $V$: $n$ time-series across time $T$
Initial model $Q_0 = m, T_k, \lambda_k, k = 1, 2...m$
Algorithm 1 ChowLiuTS
Result: Mixture of tree distributions model: Optimal topology and
        distribution of each mixture component, distribution of state variable
        and soft partition of each data point.
1 while Not converged until tolerance do
2     E-step: Compute $\gamma_k(t)$, $\Gamma_k \forall$ $k$ and $\forall$ $t$
3             Compute $P_k(t)$ and $\forall$ $t$
4     M-step: for $k \in$ 1..$m$ do
5         Compute $\lambda_k = \frac{\Gamma_k}{N}$
6         Compute tree distribution $T_k$ = CHOW-LIU-TS($P_k$(t), V)
7     end
8 end
9 return Model $Q = \{m, T_k, \lambda_k$ ∀ k ∈ [1..m]$\}$
```

2.5.1 Computational complexity

The computational complexity of finding the probability of a certain data point from a time-dependent tree distribution is $\mathcal{O}(n)$, as given in Section 4. We do this once for each tree and data point for each E-step, a total of $\mathcal{O}(mnT)$. Computation of the aggregate downstream variables like γ is the same complexity as well. The M-step involves learning m time dependent tree distributions. This is $\mathcal{O}(n^2T)$ per component, as explored in Section 4. Per iteration, the total would be $\mathcal{O}(mn^2T+mnT)$. There are no convergence results on EM when used with mixtures of time dependent trees, and thus we cannot estimate the number of iterations to converge to a solution. However, the convergence criteria is a parameter we control, and in our experiments, we found that convergence is reached on the order of 10 iterations.

2.5.2 Choosing the number of mixture components

Given a dataset with unobserved choice variable, running algorithm 2 gives us a soft partitioning at every iteration. At the end of the algorithm, we get the best labeling of the data into m clusters, where m is the number of mixture components. This is how one can use this for unsupervised learning. We can also vary m to set the number of desired clusters, and use a metric like BIC or MDL to decide the optimal number of clusters. We leave investigating this for future work.

2.6 Evaluation and experiments

This section describes the experiments we ran to validate and evaluate our implementation of both the time-dependent tree distribution and the mixture of time dependent tree distributions models.[2]

[2]Source code of the implementation, and for generating all the results presented herein, will be available at
`https://github.com/snugghash/mixtrees-for-time-series`
after peer reviewed publication.

Unless specified otherwise, training algorithms are initialized with newly generated tree distributions for each trial.

All algorithms are run until convergence, which we defined to be a change of less than 0.1% log-likelihood over an iteration. The experiments are repeated 24 times for statistical confidence. This was chosen because the cluster we ran on had 24 nodes. We use $\log_2$ based metrics (like average log-likelihood ratios between the generating and estimated distributions) unless otherwise noted.
In the graphs, mean values are generally represented in solid blue, with grey lines for each trial and transparent light blue areas for 95% confidence intervals. The normality tests for these areas have both skew and kurtosis components, and use a p-value threshold of 0.001. In case of $p < 0.001$, we plot each of trials and their mean, without the light blue area. The dots represent sampled datapoints and the lines in-between the dots represent interpolation.

2.6.1 Environment

All experiments are run with Python 3.6, on an x86 GNU/Linux machines in 24 core nodes with Intel Xeon Gold "Sky Lake" processors and 96 GB of memory. All the trials are run in parallel. The baseline tolerance for EM iteration stopping is 10^{-3}.

2.6.2 Artificial dataset: Mixture of random directed rooted tree topology

We use mixtures of tree distributions as the data generating (or “true”) distributions. We generate these models by first generating a valid directed rooted tree topology for each component (for a given number of components), then generating mixing coefficients for each component, and then generating the data using a conditional distribution for each time-series.

For each mixture component, we first generate a so-called *random minimum spanning tree* topology. A matrix of random numbers $\in (0, 1)$ is generated, which is set to be an adjacency matrix. We note that this construction does not generate a uniform distribution among all spanning trees, though every spanning tree has positive probability of being selected. Then, Edmonds-Chu is run on the complete graph described by this matrix, obtaining a maximum-weight directed rooted tree.

For the proportion of each mixture component in the model, we randomly choose lambdas (mixing proportions of the components) with a minimum of 0.1 and a maximum of 0.9 (or the amount left, whichever is smaller).

For our experiments, we use binary time-series. The $t = 0$ data point is generated with a uniform probability distribution. All later data points are generated by selecting a component according to the λ_k parameters.

For each conditional distribution to be generated, coefficients of dependence are selected at random. For root nodes, the only dependence is on their own past. Thus, one coefficient of dependence is selected $c_i \in [2, 4]$. The $Pr(V(t) = 1) = \frac{1}{5}c_i V_i(t-1) + 0.1$. For non-root nodes dependent on one more node, two coefficients are selected $\in [1, 2]$. For them, $Pr(V(t) = 1) = \frac{1}{5}c_{i1} V_i(t-1) + \frac{1}{5}c_{i2} V_j(t-1) + 0.1$. Thus, we see that the final $Pr(V(t) = 1)$s are $\in [0.1, 0.9]$, with the lower bound of c_i dictating how certain the distribution is dependent.

2.6.3 Sample complexity

We first want to measure how many samples are required to learn an effective model with all parameters, and then how quickly the model converges to the generating model with increasing samples to learn from. The distance metric to measure this convergence is the empirical log likelihood evaluated over a large (here 26.4k samples) dataset unseen during training. This can be represented as

$$\lim_{t \to \infty} \frac{1}{t} \log \frac{P_{gen}(V^{\text{test}})}{\widehat{P}(V^{\text{test}})}$$

where $\widehat{P}$ is the log-likelihood of data V^{test} using the mixture model distribution inferred on the training data V^{train}. This average log likelihood ratio's expected value and its empirical limit is the Kullback-Liebler divergence (Ch 3 of [31]). We refer to this quantity as "Testing Error." We expect this quantity should be positive (the statistical mean, namely the Kullback Liebler divergence, is non-negative, though it is possible for the empirical mean to be negative).

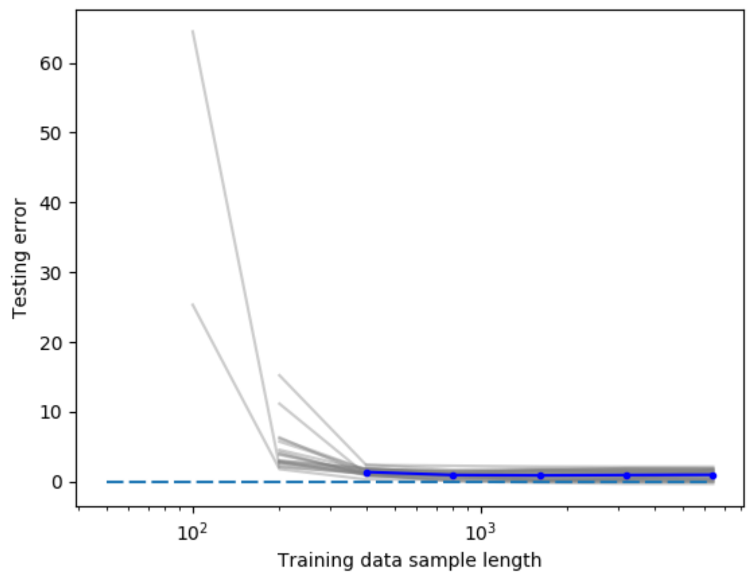
We explore how the testing error varies as the number of training samples T^{train} increases. Results are depicted in Figure 2.5. The grey lines depict each of the 24 trials, and the blue line depicts the mean and one standard deviation above and below the mean. There is a large drop in error initially, and then the error stabilizes. The figure suggests that although the divergence does not go to zero in the range shown in the figure, that by T^{train} the improvement in using up to triple that amount is marginal.

2.6.4 Classification

We now seek to investigate how well components of the mixture distributions can be learned. We conduct a classification experiment with using a generative mixture of trees distribution. The task is to correctly predict which component a particular datapoint came from, within the generative mixture distribution. We test the abilities of the learning algorithm against the best-in-class distribution, the generating distribution itself. We keep the number of components to two. The training and test datasets are separate 1000-length datasets. Since the two distributions are of the same class, we expect that the learned model should perform just as well as the generating distribution with enough training data, and the area under curve should be approximately the same. We expect the size of this "large" dataset where the two converge to be a few hundred datapoints, from Figure 2.5.

The resulting ROC curve is shown in Figure 2.6. As expected, the generative model does a better job of classifying which component individual test data points came from, though the fact that the area under the curve is far from 1 indicates the intrinsic challenge of this task.

Remark 2 *The two tree components of the generative model had randomly selected topologies and conditional dependencies. We conjecture that in the limit that both*



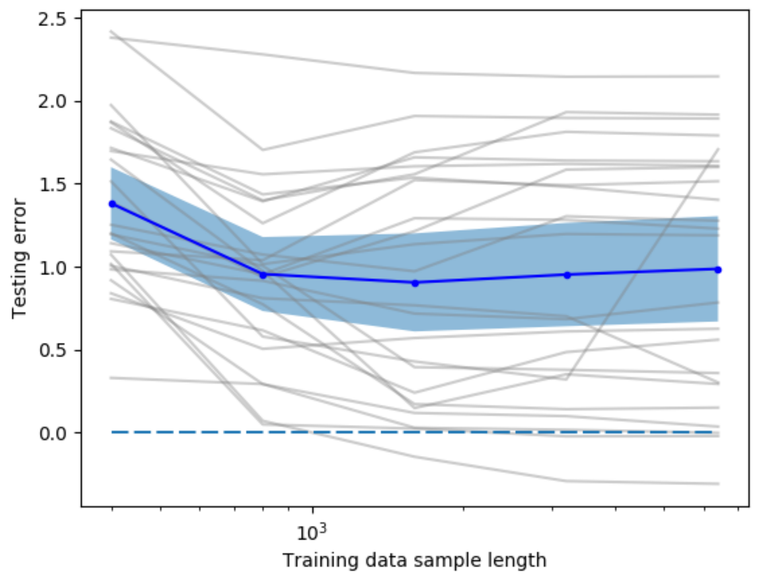


Figure 2.5. Testing the divergence to the true model (comparing likelihood over the same large testing dataset of 26.4k samples) vs. increasing number of samples given to learn. Learning of the model is performed over a smaller number of samples (x-axis) and the y-axis is the log likelihood ratio evaluated over an unseen larger test dataset. The bottom figure is a zoomed-in snapshot of the top figure.

components are almost identical (such as in Kullback Liebler divergence or in total variation), then even the generative model would perform the same as random guessing. Likewise, we anticipate that example mixture of trees generative distributions could be constructed with the underlying components quite distinct (large divergence) enabling the generative distribution and the learned distribution to easily distinguish.

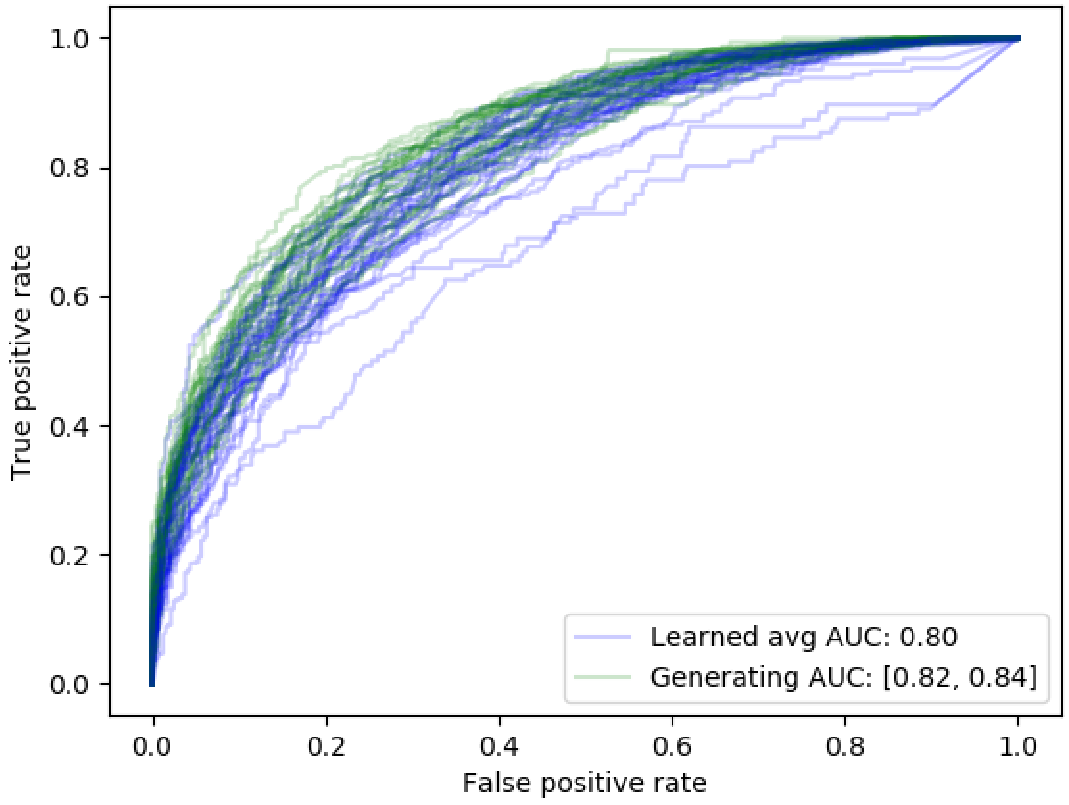


Figure 2.6. Classification as compared to the generative model. The generative distribution performs at the same level as the learned model. The learned model AUC failed normality test, the mean alone is shown.

2.6.5 Clustering

Given a dataset, we can train a mixture of time-dependent trees model on it, giving us soft partitioning at every step (with weights $\gamma_k(t)$ for time-point $V(t)$ coming from component k). If we quantize the soft partitioning to a hard partitioning, this is effectively clustering of the dataset into m clusters where m is the number of components. We want to measure performance of this implementation at clustering data generated from its own class. We use the metric of adjusted Rand Index (aRI) to do this, which if perfect will be 1, and will be 0 when the model performs the same as a uniform random clustering.

A generative model was randomly selected, and 24 trials of learning on the same dataset with different seed are performed. An expectation step is performed with each of the learned models and the one generative model, giving us a soft partitioning for each. Along with the true labels, we can now compute the aRI. We expect our results to be close to that from the generative distribution, except for convergence issues. The number of components is 2. Amount of data used is 10000 samples. Note that the training and testing dataset is the same for this experiment.

We observe that the generative distribution performs poorly itself, showing that this task is intrinsically hard. The learned model performs significantly worse, with high variation among trials. This is surprising, considering the results from classification. The variance is only from the randomized seed models for EM, which shows a large number of trials getting stuck in local optima, as expected. Also as expected, these are all still significantly better than randomness (which is aRI=0).

2.6.6 Runtime scaling

Characterizing the runtime scaling of this model is performed over varying training samples, number of time series, and number of components. The baseline hyperparameters are 10000 training datapoints, 10 time series, and 2 components. The CPU time within each thread is measured in seconds, thus giving us comparable results

Table 2.1.
Clustering of a dataset into the most likely component for each datapoint. This is measured with (chance) adjusted Rand Index, 0 is same performance as a random clustering, and 1 is perfect. The same dataset and generative model is used, with the EM seed model being the only variable across trials.

Learned model aRI	Generating model aRI
(0.16, 0.23)	0.24

across trials. The parallelism with distributed memory ensured no caching takes place, which might speed up later trials.

Building on the computational complexity calculations in subsection 2.5.1, we expect the complexity with respect to the number of time-series n to be quadratic, linear with samples T.

Table 2.2 illustrates the results, with normal statistics. We see slightly above-quadratic scaling with time-series from the data, consistent with expectations. We expect linear scaling with number of samples T, and we see it only in the higher sample range - this is consistent with a significant static overhead, that becomes less of a contributor with more samples. Finally, we expect polynomial scaling with low number of mixture components, and the results are consistent with that interpretation, although we need to test this with high number of components to be certain.

2.6.7 Trade-offs with changing tolerance

The criteria we use for stopping EM is a percentage change in the log-likelihood over the training data, which is a hyperparameter that we can tune. These results should be domain-specific, since different estimators can give different convergence characteristics. The baseline parameters are 10 time-series, 2 components, 1k samples for training and 26.4k samples for testing.

Table 2.2.
Runtime scaling with number of time series, samples, threshold, number of mixture components. All times are expressed in seconds. When not varying, the parameters are kept constant at 10 time-series, 2 components, 10000 samples.

Number of time series	Runtime (95% over 24 trials)
5	(29, 56)
10	(91, 131)
20	(586, 946)
40	(2092, 3299)

Samples	Runtime (95% over 24 trials)
100	(10.1, 11.1)
400	(13.9, 17.3)
1000	(21.0, 27.4)
2500	(39.3, 63.7)
5000	(69.4, 113.3)
10000	(105.2, 182.0)

Components	Runtime (95% over 24 trials)
1	(19.4, 19.6)
2	(227, 300)
3	(283, 477)
4	(725, 1182)
5	(1443, 2440)

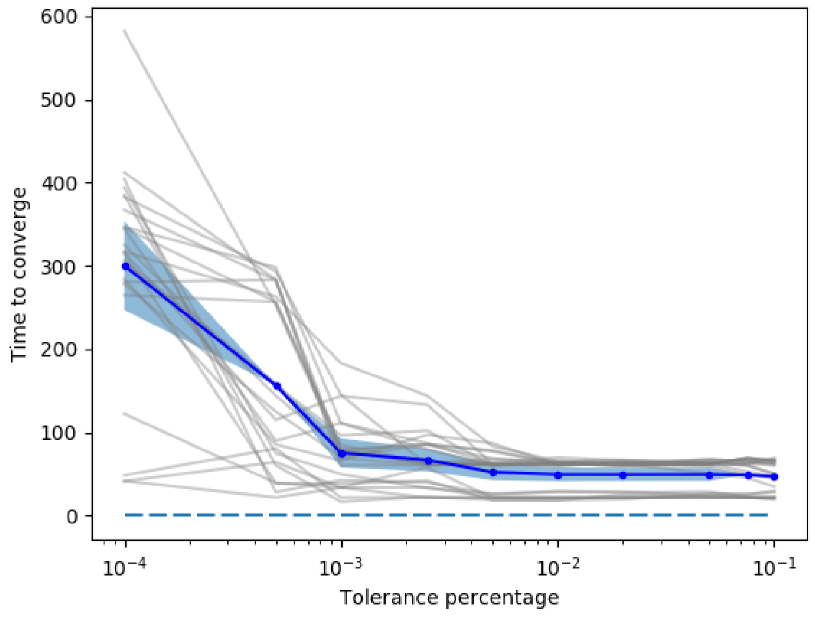


Figure 2.7. Learning time in seconds plotted against changing tolerance fraction.

In Figure 2.7, we plot the run-time as a function of the tolerance for iteration stopping in EM. We expect polynomial or worse scaling with increasing tolerance, since we expect diminishing returns with more iterations of EM. We notice a steep decrease until 10^{-3}, and then a flattening out. This is suggestive of a large number of iterations EM being required for gains of 10^{-3} in log-likelihood. It also shows us that a good tolerance level for this environment is at 10^{-3}.

We also investigate the divergence between the generative model and the learned model as a function of the EM iteration tolerance. This is computed as in subsection 2.6.3, with 26.4k samples serving as the large testing dataset. We generally anticipate that a smaller tolerance will lead to solutions for the model parameters closer to a locally optimal solution. In the large training sample limit, smaller tolerances consequently should have smaller divergence.

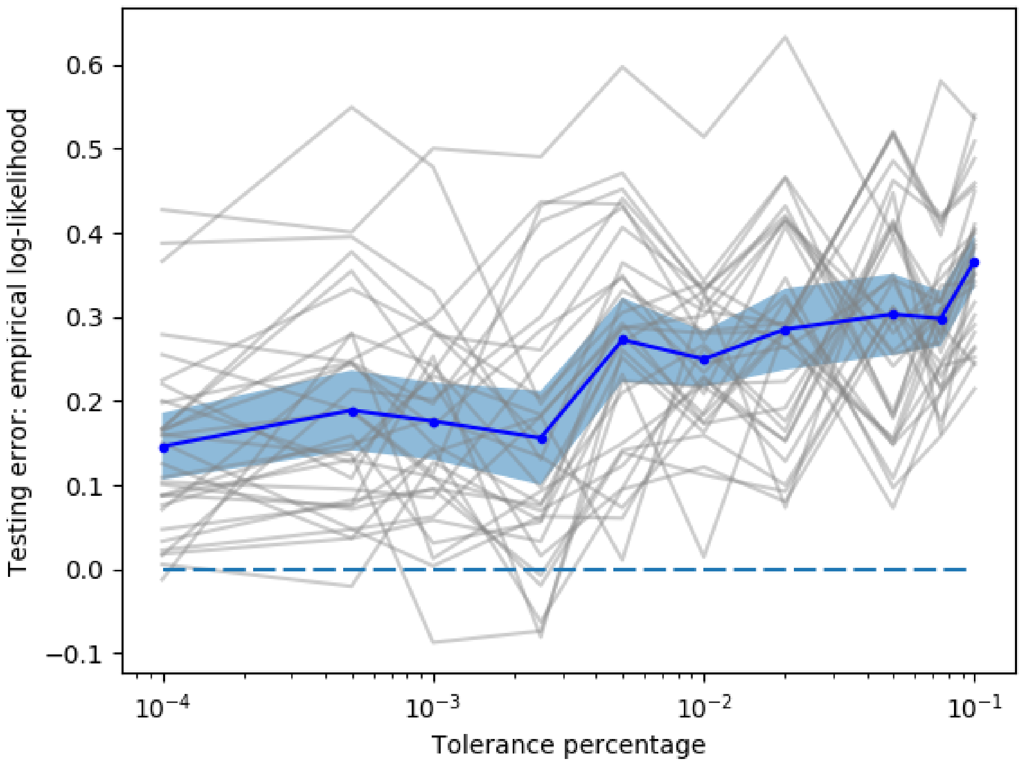


Figure 2.8. Divergence to generating distribution over testing dataset, plotted against tolerance.

From the results in Figure 2.8 we observe that there is indeed lower divergence (better learned model) with the lower tolerances, and a trend of increasing divergence as the tolerance increases. However, there is a drop and spike in divergence close to a tolerance of 0.075; we are unsure of the source of this.

3. MIXTURES OF TREES WITH STATE ESTIMATION

3.1 Introduction

Consider the rainfall prediction task from the last chapter. The dependencies across the region of weather stations might change in a predictable way, across the seasons, the day-night cycle. We consider the case where this change of distribution can be modeled (or approximated) as a hidden Markov model.

Our goal in this chapter is to develop formulation and learning algorithms for hidden Markov models with autoregressive observables with tree dependencies. Put another way, these are mixtures of time-dependent trees models where the latent state or hidden choice variable follows a Markov process. An illustration of how evolving state might influence the dependencies in variables is Figure 3.1. Note that if the states were i.i.d, we would have the situation in the last chapter, described in Section 2.4.

We begin with a description of the classical hidden Markov model in Section 3.2, and formulate the relevant learning algorithm, one based on the Baum-Welch Section 3.3 [9]. We then extend this to work with time-dependent tree dependencies among observed variables Section 3.4.

3.2 Hidden Markov models

Hidden Markov models (HMMs) are characterized by a set of observed variables whose probability distributions are dependent on a hidden state variable at that time step, $Z(t)$, which itself is a random variable following a Markov process. Let the

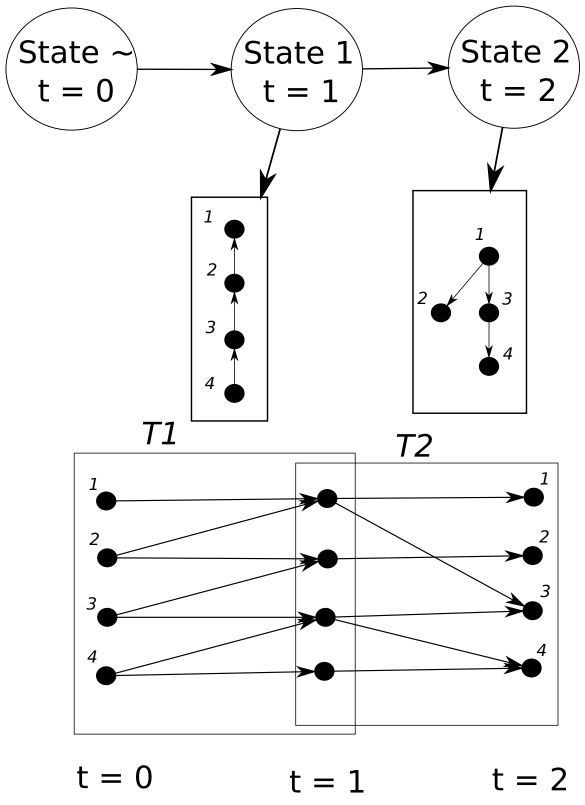


Figure 3.1. An illustration of how hidden states affect the dependencies of variables, and how the states are time-dependent Markov chains themselves.

states of Z be from the set $\Omega_z = \{1, 2, 3 \ldots m\}$. Denote the time-homogeneous *state transition matrix* of Z with S:

$$S := \begin{bmatrix} S_{1,1} & S_{1,2} & \ldots \\ S_{2,1} & S_{2,2} & \ldots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{3.1}$$

so

$$S_{i,j} = P(Z(t) = j \mid Z(t-1) = i),$$

the probability that the hidden Markov process in state i transitions to state j in the next time step. Let π denote the intial distribution of the hidden state,

$$\pi_k := P(Z(1) = k).$$

As in the i.i.d. case, the observed variables are modeled as coming from a mixture of trees distribution, denoting the kth mixture component distribution as

$$T_k(V(t), V(t-1)) = P(V(t)|V(t-1), Z(t) = k).$$

Note that the dependence of $V(t)$ on $V(t-1)$ given the hidden state $Z(t)$ is in contrast to the standard hidden Markov model assumptions, for which the Baum-Welch algorithm was proposed.

3.3 Modified Baum-Welch algorithm for HMMs

We will use an iterative EM approach for this setting analogous to that for the i.i.d. setting described in Section 2.5. The derivations are similar here, with the main difference being that the hidden state depends on its own past. We will present the derivations of the E and M steps more concisely than in Section 2.5, highlighting the differences with the i.i.d. setting. We will see in particular that the E-step is more challenging, and we need to use a forward-backward procedure to calculate the posterior. In the literature for HMMs with i.i.d observables, this variant of EM is called the Baum-Welch algorithm [33]. Given an initial model, we recursively compute the most likely sequence of states (using dynamic programming to make this tractable) until we get a soft partitioning i.e. the probability of each data point coming from each state. This is called the expectation (E) step. We then update our model to maximize expected likelihood with respect to data this new partitioned data, thus getting a model with better likelihood estimate than we started with. In contrast to the standard Baum-Welch algorithm for hidden Markov models, here the emitted variables, namely $\{V(t)\}$, are *not* conditionally independent given the hidden states. This case was investigated in [23] and a modified version of the Baum-Welch was proposed for linear auto-regressive time-series. Our derivation is largely the same as [23], except that here the emission distributions are mixtures of tree distributions and our approach is not restricted to the linear setting.

We can express the *complete log likelihood* as

$$\begin{aligned}
&l_{complete}(V, Z \big| Q(V, Z)) \\
&\quad = \log Q(V, Z) \\
&\quad = \log \prod_{t=1}^{T} Q(V(t), Z(t)|V(t-1), Z(t-1), \dots) \\
&\quad = \sum_{t=1}^{T} \log Q(V(t), Z(t)|V(t-1), Z(t-1)) \\
&\quad = \sum_{t=1}^{T} \log Q(Z(t)|V(t-1), Z(t-1))Q(V(t)|V(t-1), Z(t-1), Z(t)) \\
&\quad = \sum_{t=1}^{T} \log Q(Z(t)|Z(t-1))Q(V(t)|V(t-1), Z(t)) \\
&\quad = \sum_{t=1}^{T} \log Q(Z(t)|Z(t-1)) + \log Q(V(t)|V(t-1), Z(t)) \\
&\quad = \sum_{t=1}^{T} \log S_{Z(t-1),Z(t)} + \log T_{Z(t)}(V(t), V(t-1)) \\
&\quad = \sum_{t=1}^{T} \sum_{i=1}^{m} \sum_{j=1}^{m} \delta_{i,Z(t-1)}\delta_{j,Z(t)} \left(\log S_{i,j} + \log T_j(V(t), V(t-1))\right) \qquad (3.2)
\end{aligned}$$

We will again take the expectation of this complete log likelihood with respect to the conditional distribution $Q(Z|V)$,

$$E_{Q(Z|V)}\left[\, l_{complete}(V,Z \mid Q(V,Z)) \,\middle|V\right]$$
$$= \sum_{t=1}^{T}\sum_{i=1}^{m}\sum_{j=1}^{m} E_{Q(Z|V)}\left[\delta_{i,Z(t-1)}\delta_{j,Z(t)}\middle|V\right]\left(\log S_{i,j} + \log T_j(V(t),V(t-1))\right). \quad (3.3)$$

In the M-step, this function will be maximized over distributional parameters. The E-step will calculate

$$\begin{aligned} &E_{Q(Z|V)}\left[\delta_{i,Z(t-1)}\delta_{j,Z(t)}\middle|V\right] \\ &\quad = P\big(Z(t-1)=i, Z(t)=j\big|V\big) \\ &\quad = \frac{P\big(Z(t-1)=i, Z(t)=j, V\big)}{P\big(V\big)} \\ &\quad = \frac{Pr\big(Z(t-1)=i, Z(t)=j\big)P\big(V|Z(t-1)=i, Z(t)=j\big)}{P\big(V\big)} \\ &\quad = \frac{P(Z(t)=j \mid Z(t-1)=i)P(Z(t-1)=i)}{P(V)} \\ &\qquad \times P(V(1),V(2)\ldots V(t-1)\big|\; Z(t-1)=i) \\ &\qquad \times P(V(t)\big|Z(t)=j, V(t-1)) \\ &\qquad \times P(V(t+1)\ldots V(T)\big|Z(t)=j, V(t)) \end{aligned}$$

We next introduce notation to simplify the above expressions and facilitate calculating the above expression. Define the two parts of the numerator:

$$\alpha_k(t) = P(V(1),V(2),...V(t),Z(t)=k) \quad (3.4)$$
$$\beta_k(t) = P(V(t+1),V(t+2),...V(T)\big|Z(t)=k,V(t)) \quad (3.5)$$

The quantity $\alpha(T)$ represents the joint probability of observing all of the given data up to time T and having a final hidden state $Z(T)$.

We can use these parameters in evaluating the conditional expectation of the log likelihood,

$$\begin{aligned}
&E_{Q(Z|V)}\left[\delta_{i,Z(t-1)}\delta_{j,Z(t)}\middle|V\right]\\
&\quad=\frac{P(Z(t)=j\mid Z(t-1)=i)P(Z(t-1)=i)}{P(V)}\\
&\quad\times P(V(1),V(2)\dots V(t-1)\big|\ Z(t-1)=i)\\
&\quad\times P(V(t)\big|Z(t)=j,V(t-1))\\
&\quad\times P(V(t+1)\dots V(T)\big|Z(t)=j,V(t))\\
&=\frac{S_{i,j}\times\alpha_i(t-1)\times T_j(V(t),V(t-1))\times\beta_j(t)}{P(V)}.
\end{aligned}\tag{3.6}$$

We now derive recursion relations for computing α and β efficiently, in order to calculate (3.6).

$$\begin{aligned}
\alpha_k(t)&=P(V(1),V(2),\dots V(t),Z(t)) &(3.7)\\
&=\sum_{i=1}^{m}P(V(1),V(2),\dots V(t-1),Z(t-1)=i,V(t),Z(t)) &(3.8)\\
&=\sum_{i=1}^{m}P(V(1),V(2),\dots V(t-1),Z(t-1)=i)\\
&\quad\times P(V(t),Z(t)\mid V(1),V(2)\dots V(t-1),Z(t-1)=i) &(3.9)\\
&=\sum_{i=1}^{m}\alpha_i(t-1)P(Z(t)\mid V(1),V(2)\dots V(t-1),Z(t-1)=i)\\
&\quad\times P(V(t)\mid V(1)\dots V(t-1),Z(t-1)=i,Z(t)) &(3.10)\\
&=\sum_{i=1}^{m}\alpha_i(t-1)\times S_{i,Z(t)}\times T_{Z(t)}(V(t),V(t-1)). &(3.11)
\end{aligned}$$

α will converge to 0 for large t. To mitigate this, we will compute normalized α's,

$$\widehat{\alpha}_k(t-1)=\frac{\alpha_k(t-1)}{\sum_k\alpha_k(t-1)}\tag{3.12}$$

Starting from an initial estimate of α at time 0, using the known transition probabilities S and emission probabilities $T_{k}{}_{k=1}^{m}$, we can compute α for all time. This is the "forward step" for calculating $P\big(Z(t-1)=i,Z(t)=j\big|V\big)$. Considering that we're evaluating $\alpha(t)$ for all K components (one for each possible state), the computational cost of each time step scales as $\mathcal{O}(m^2)$. Next, we identify a recursion relation for the β terms,

$$\begin{aligned}
\beta_k(t) &= P(V(t+1), V(t+2), ...V(T) \mid Z(t), V(t)) && (3.13)\\
&= \sum_{i=1}^{m} P(V(t+1), V(t+2), ...V(T), Z(t+1) = i \bigm| Z(t), V(t)) && (3.14)\\
&= \sum_{i=1}^{m} P(V(t+1), V(t+2), ...V(T) \bigm| Z(t+1) = i, Z(t), V(t)) && (3.15)\\
&\quad \times P(Z(t+1) = i \bigm| Z(t), V(t)) && (3.16)\\
&= \sum_{i=1}^{m} P(V(t+1), V(t+2), ...V(T) \bigm| Z(t+1) = i, V(t)) && (3.17)\\
&\quad \times P(Z(t+1) = i \bigm| Z(t)) && (3.18)\\
&= \sum_{i=1}^{m} P(V(t+1) \bigm| Z(t+1) = i, V(t)) \\
&\quad \times P(V(t+2), V(t+3), ...V(T) \bigm| Z(t+1) = i, V(t), V(t+1)) && (3.19)\\
&\quad \times P(Z(t+1) = i \bigm| Z(t)) && (3.20)\\
&= \sum_{i=1}^{m} T_i(V(t), V(t+1)) \times \beta_i(t+1) \times S_{Z(t),i} && (3.21)
\end{aligned}$$

This is the "backward" procedure, with a computational complexity of $\mathcal{O}(m^2)$ per time step. This starts at $t = T$ and recursively computes β back through time. We initialize $\beta_k(T) = 1$. As with α, we normalize β terms to avoid numerical issues.

Note that using these parameters, the the likelihood function $P(V)$ has a simple expression. For any t,

$$\begin{aligned} P(V) &= \sum_{k=1}^{m} P(V, Z(t) = k) \\ &= \sum_{k=1}^{m} P(V(1), \ldots, V(t), Z(t) = k) P(V(t+1), \ldots, V(T) | V(t), Z(t) = k) \\ &= \sum_{k=1}^{m} \alpha_k(t) \beta_k(t) \qquad (3.22) \\ &= \sum_{k} \alpha_k(T) \qquad (3.23) \end{aligned}$$

As at $t = T$, β is a vector of 1s.

Now, computing the posterior probability γ (also known as *smoothing step* in the literature):

$$\gamma_i(t) = P(Z(t) = i \big| V) = \frac{P(Z(t) = i, V)}{P(V \big| \theta)} = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{m} \alpha_j(t)\beta_j(t)} \qquad (3.24)$$

$$\begin{aligned} \eta_{ij}(t) &= P(Z(t-1) = i, Z(t) = j \mid V) \qquad (3.25) \\ &= \frac{S_{i,j}\alpha_i(t-1) T_j\big(V(t), V(t-1)\big)\beta_j(t)}{P(V)}. \qquad (3.26) \end{aligned}$$

Where P(V) is from Equation 3.23.

Next, we re-write the conditional expectation of the complete log likelihood (3.3),

$$\begin{aligned} & E_{Q(Z|V)} \left[l_{complete}(V, Z \mid Q(V, Z)) \middle| V \right] \\ &= \sum_{t=1}^{T} \sum_{i=1}^{m} \sum_{j=1}^{m} E_{Q(Z|V)} \left[\delta_{i,Z(t-1)} \delta_{j,Z(t)} \middle| V \right] (\log S_{i,j} + \log T_j(V(t), V(t-1))) \\ &= \sum_{t=1}^{T} \sum_{i=1}^{m} \sum_{j=1}^{m} \eta_{i,j}(t) (\log S_{i,j} + \log T_j(V(t), V(t-1))) \qquad (3.27) \end{aligned}$$

The E-step is about efficiently computing γ and η. This is done in terms of α and β. This E-step gives us two things - the $l_{complete}$ which is the fitness, and the probability that each time point was generated by a given state $P(Z(t) = z|V)$. This is considered to be a soft partitioning of the data. Next, in the M-step, we treat γ

and η as constants to maximize the likelihood by changing θ, the parameters of the model, fitting it to the *partitioned* data, each partition describing one state's emission distribution. The analytical solution relies on Lagrange multipliers, yielding updates to the initial state distribution and state transition probabilities:

$$\pi_k = \frac{\gamma_k(1)}{\sum_{j=1}^{m} \gamma_j(1)} \tag{3.28}$$

$$S_{i,j} = \frac{\sum_{t=2}^{T} \eta_{i,j}(t)}{\sum_{t=2}^{T} \sum_{l=1}^{m} \eta_{i,l}(t)}. \tag{3.29}$$

3.4 Emission probabilities update for time dependent tree distributions as observables

In the M-step, we update on the distributions $\{T_k\}$ by obtaining new estimates of each of the probability distributions above, and then running the Chow-Liu directed algorithm to obtain the best time-dependent tree distribution.

Now, $\gamma_k(t)$ is defined as the likelihood of data at time t being generated from hidden state k. In the M-step, this provides a soft partitioning of the data. When updating (learning) the corresponding time-dependent tree distribution corresponding to state k, each datapoint in time can be viewed as being weighted by this probability. Using this modified dataset as the basis, we can use our estimator for each P (our implementation uses an empirical distribution) to obtain the new time-dependent tree distribution, as described in subsection 2.3.3.

In summary, the EM algorithm also described in algorithm 3 requires us to make an initial selection of parameters $\theta \equiv (\pi, S, E)$. The S and E distributions can be either uniform or random (if the landscape is such that it often gets stuck in local maxima, we are better off using multiple trials of random initialization). Given the dataset V, we run the forward and backward steps to compute α and β and consequently compute γ, η the complete log likelihood. This is the E-step. We now use these results to change the parameters of our model to θ^{new}. We continue to alternate E and M steps until we reach a convergence criteria, typically one of small change in likelihood computed in the E-step.

Algorithm 3: Baum Welch with autoregressive tree dependencies in observables

```
Data: $n$ variables across time $T$
Initial model $Q_0 = \pi, S, T_k$
Algorithm 1 Learn-timeSeries-tree
Result: Hidden Markov model with time dependent tree distributions:
        Optimal structure and parameters of each states' emission
        distribution, state transition distribution, and soft partitioning of
        each data point
1  while Not converged until tolerance do
2      E-step: Compute $\gamma_k(t)$, $P_k(t)$ $\eta_{(k,j)}(t)$, likelihood $\forall$ $k \in m$ and $\forall$ $t \in [1..T]$
3      M-step: for $k \in 1..m$ do
4          Compute $\pi_k$
5          for $j \in 1..m$ do
6              Compute $S_{j,k} \forall$ states $j, k$
7          end
8          $T_k \leftarrow$ ChowLiuTS($P_k$(t), data)
9      end
10 end
11 return Model $Q = \{m, T_k, S_{(k,j)}, \pi_k$ $\forall$ k $\in$ [1..m]$\}$ and final partitioning
     $\gamma_k(t) \forall t \in [1..T]$
```

3.4.1 Computational complexity

The forward step of the forward-backward algorithm involves two passes over all the states, and one pass over all the dependencies (for computing likelihoods) for every time point, thus is of order $\mathcal{O}(m^2nT)$ steps. The backward step requires the same number of passes over each of the hyperparameters, bringing the total of the E-step to $\mathcal{O}(m^2nT)$. From Chapter 2, Section 4, learning of time dependent tree distributions comes out to be $\mathcal{O}(n^2(T + \left|\Omega\right|^3))$ or $\mathcal{O}(n^2T)$, based on implementation. Then, the M-step, learning m different emission distributions is $\mathcal{O}(mn^2(T+\left|\Omega\right|^3))$ or $\mathcal{O}(mn^2T)$. Finally, the full algorithm is of $\mathcal{O}(mn^2(T + \left|\Omega\right|^3) + m^2nT)$ or $\mathcal{O}(mn^2T + m^2nT)$, assuming a constant number of iterations of EM is required to converge.

3.5 Evaluation

3.5.1 Environment

We will use the same computational setup as in the i.i.d. hidden state case (subsection 2.6.1).

All the trials are run in parallel. The default parameters are 10 time-series, 1000 training samples, and 26.4k samples for computing log-likelihood on a large dataset (testing error or divergence). The baseline tolerance for EM iteration stopping is 10^{-3}.

3.5.2 Artificial data: Random state distribution, random TD tree distribution

We use Markov hidden state with time dependent tree emission distributions as the data generating distribution.

Generation of the emission distribution (the time-dependent trees) is done with the same process as in Chapter 2, subsection 2.6.2. Instead of generating valid mixing coefficients that sum to 1, we now generate a state transition matrix. We uniformly choose entries in the matrix with the same limits of $(0.1, 0.9)$ to prevent stable attractors in the hidden Markov state. Then, we generate a time-series for the hidden-state, with the initial distribution being uniformly random.

For our experiments, we use binary-alphabet time-series. The $t = 0$ data point is generated with a uniform probability distribution. Based on the state variable governing the relationship between the observable time-series, each future time-point is generated with the corresponding emission conditional distribution as in the i.i.d state scenario.

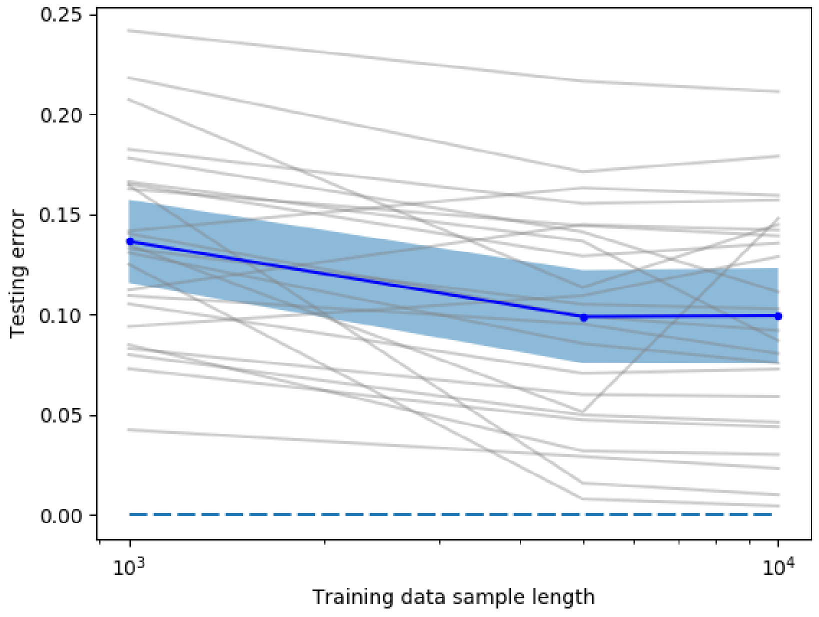
3.5.3 Sample complexity

The first experimental metric we want to measure is how many samples are required to learn an effective model with all parameters, and then how quickly the model converges to the generating model with increasing samples to learn from. The metric to measure this convergence is the empirical log likelihood evaluated over a large (here 26.4k samples) dataset unseen during training, same as the i.i.d state case subsection 2.6.3. Analogously, we also expect this error to be positive, since generative model will be better than the learned model over an unseen dataset. We expect it to decrease with increasing training samples, asymptotically going to 0, as more of the parameters converge to those the generating model. To get better comparisons, we use the same seed model across all the sizes of training samples, since different EM starting seeds can converge to different local optima, skewing the results.

The results are shown in Figure 3.2. There is a large error when using very low samples. This is consistent with there being not enough samples to adequately learn the model parameters in that regime. We also notice a fairly flat regime beyond 1000 samples, where the marginal improvement from more data gets smaller and smaller. This is also consistent with the expectation of asymptotic convergence to generating model.

3.5.4 Clustering

We now use the soft-partitioning from the EM algorithm to label each training data point, clustering it into one of the components. Similar to the i.i.d state case



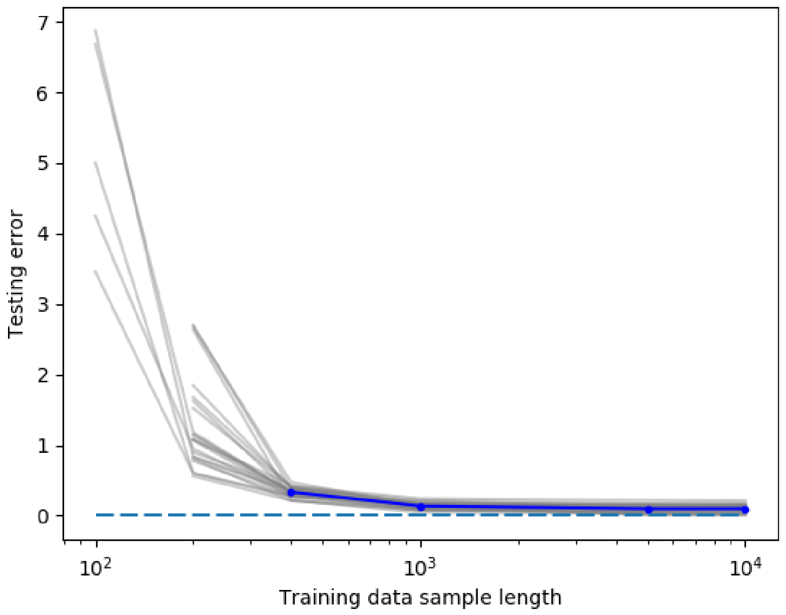


Figure 3.2. Testing the divergence to the true model (likelihood ratio over 26.4k samples) vs. increasing number of samples given to estimate. The grey lines represent each of the 24 trials, each with the same generative distribution and dataset. The blue line is the mean, with the blue region being 95% confidence regions.

in subsection 2.6.5, we hope to evaluate the unsupervised learning capabilities of the model. We use the adjusted (for chance) Rand Index metric, which is 0 for a model performing same as chance and 1 for the perfect clustering. We choose 1000 training samples, 2 components, and 10 times series. To be able to gather insights, we keep the generative model and the dataset same across trials, since the generative aRI changes significantly across trials.

Analogous to the i.i.d state case, we expect the generative model to be mediocre, and the learned model to be somewhat worse. The results are shown in Table 3.1. As expected, both the generative and learned model aren't too great, with the learned model being significantly worse than chance. We note that clustering of the training samples might be a particularly hard task, but we are unsure of the root cause for why the learned model performs so much closer to the generative model on the similarly set-up classification task.

Table 3.1.
Clustering of a dataset into the most likely component for each datapoint. This is measured with (chance) adjusted Rand Index, 0 is chance, 1 is perfect. The normality test is satisfied.

Learned model aRI	Generating model aRI
(0.11, 0.22)	0.34

3.5.5 Classification

We conduct a classification experiment with using a generative HMM model with time-dependent tree emission distribution, to investigate how well the components are being learned. The task is to correctly predict which component a particular unseen test datapoint came from, within the generative mixture distribution. The learning model is the same class as the generating model, both with two components. The training and test datasets are separate 1000-length datasets. Analogous to the i.i.d case, we expect that the learned model should perform as well as the generative model with enough training data, and the area under curve should be approximately the same. We expect the good ”enough” training dataset to be of size 1000, from Figure 3.2.

The resulting ROC curve is shown in Figure 3.3. As expected, the task is hard, with the generative model being far from perfect. The learned model is very close to the generative model in terms of area under the ROC curve. This leads us to conclude that the learning algorithm is successfully learning to distinguish the components from the given amount of data.

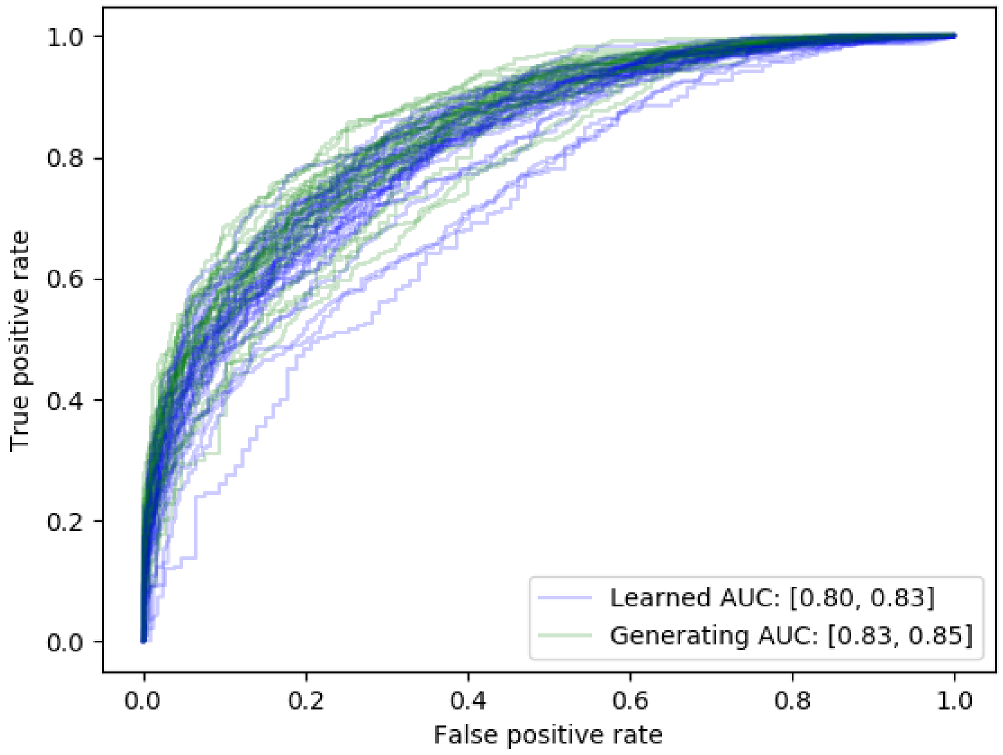


Figure 3.3. ROC and raw area under curve values for classification using Markov state TD-trees, each label is one TD-tree

3.5.6 Runtime scaling

The runtime scaling of this model is verified by learning over varying training samples, number of time series, and number of components. The baseline hyperparameters are 10000 training datapoints, 10 time series, and 2 components. From the computational complexity calculations in subsection 3.4.1, we expect the time-series complexity to be quadratic, polynomial scaling with number of mixture components, and linear in number of samples.

Table 3.2 illustrates the results, with normal confidence intervals, trained a on 10000 sample baseline. We also plot training time for 1000-samples over varying number of time-series in Figure 3.4. The learning times are linear with time-series at high number of time-series, and we conjecture that this is likely because of static overhead dominating the learning times at low number of time-series. The samples also show similar overhead behaviour, with the scaling from 5000 to 10000 samples being close to linear unlike the smaller sample regime. Learning time scales polynomially with components, also as we expect.

Remark 3 *Since the worst scaling is seen to be with increasing number of components, increasing to arbitrary components leads to intractably large runtimes.*

3.5.7 Changing number of components

Expressive power of markov state tree observables with varying number of components is an interesting metric to measure, since we can then choose the number of components parsimoniously. We select a generative model and use varying amount of components to learn from the generated dataset. Then, these models are evaluated on testing error as in subsection 2.6.3 over a large dataset generated from that generative model.

The generative distribution has 2 components. 1000 training samples were given, with a 26.4k-length unseen testing dataset over which the generating and learning

Table 3.2.
Run-time scaling with number of time series, samples, threshold, number of mixture components. The constant values are 10000 samples, 10 time series, 2 components.

Number of time-series	Runtime (95% over 24 trials)
5	(19.4, 20.6)
10	(197, 301)
20	(378, 717)
40	(782, 1074)

Samples	Runtime (95% over 24 trials)
100	(15, 18)
400	(20, 27)
1000	(35, 49)
2500	(64, 84)
5000	(130, 228)
10000	(222, 381)

Components	Runtime (95% over 24 trials)
1	Mean=50, normal test failed
2	(230, 293)
3	(555, 736)
4	(976, 1230)
5	(1700, 2120)

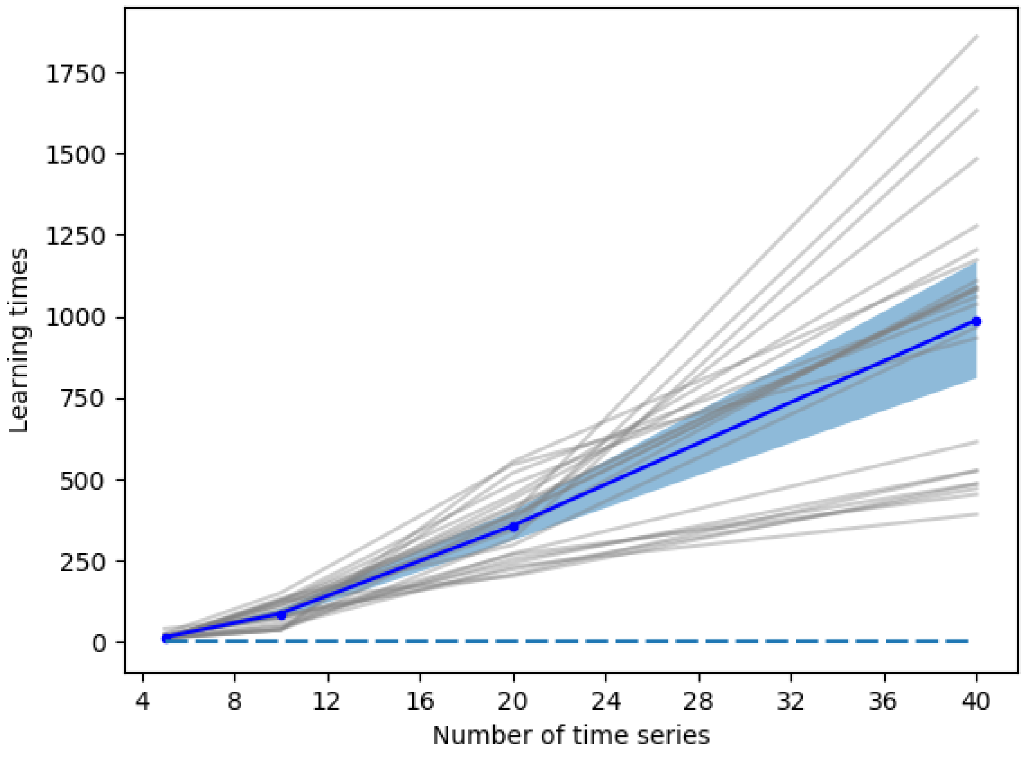


Figure 3.4. Learning time in seconds plotted with increasing number of time-series. The model had 2 components and used a 1000-long training dataset.

model are evaluated. We expect the best results for 2 component learning model, with all other components being worse. 1 component models will not have enough parameters to adequately learn the features of the dataset, and 3 and 4 component models might overfit their parameters to the training data, and thus perform much worse on the test dataset.

The results are shown in Figure 3.5. As expected, the ”elbow” in the graph occurs at 2 components. We expected 3 and 4 component models to be worse, but they are surprisingly worse than the single component case as well.

Remark 4 *This also allows us to compare the model against one of the alternatives before this work, the best single-tree approximation (for which x-axis value is 1). We observe that the model performs better (in terms of testing error) at the generative number of components, as an improved HHM model with time-dependent emission distributions.*

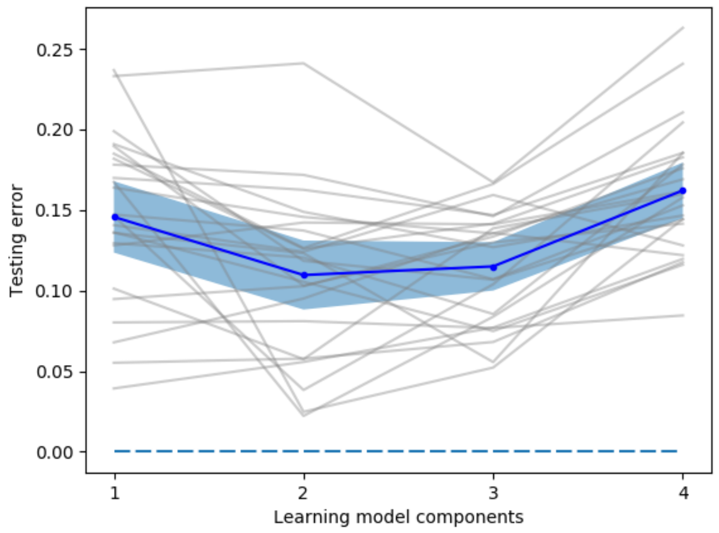


Figure 3.5. Divergence to a generative 2 component mixture vs. increasing number of learning components. The likelihoods are computed on 26.4k samples, and the model is trained on 1k samples.

3.5.8 Trade-offs with changing tolerance

Another tunable hyperparameter is the fractional change in log-likelihood at which we stop the EM iterations. We do not want to have this too low, in which case the EM would take too long to converge. At the same time, a too high tolerance would end the iteration before most of the learning is done. This tradeoff would likely be domain specific.

We measure testing error as defined in subsection 2.6.3, with a 26.4k-long testing dataset. We expect a smooth decrease as we tighten (lower) tolerance. Unexpectedly, the divergence over tolerance isn't smooth, but this is analogous to the i.i.d case at Figure 2.8. Also similarly, we observe a spike at tolerance of 0.075. We conjecture that this could be a combination of parameters that leads to the log-likelihood being trimmed particularly early.

We also measure learning times with varying tolerance, by measuring CPU time in seconds, while accounting for concurrency. We expect learning times to explode as tolerance is tightened, and for EM to never converge at some low value. The results are in Figure 3.7, and while we don't see a massive explosion in learning times, we do see an increase of about 4 times.

Remark 5 *These results also justify the baseline we use for all other experiments, of* 10^-3.

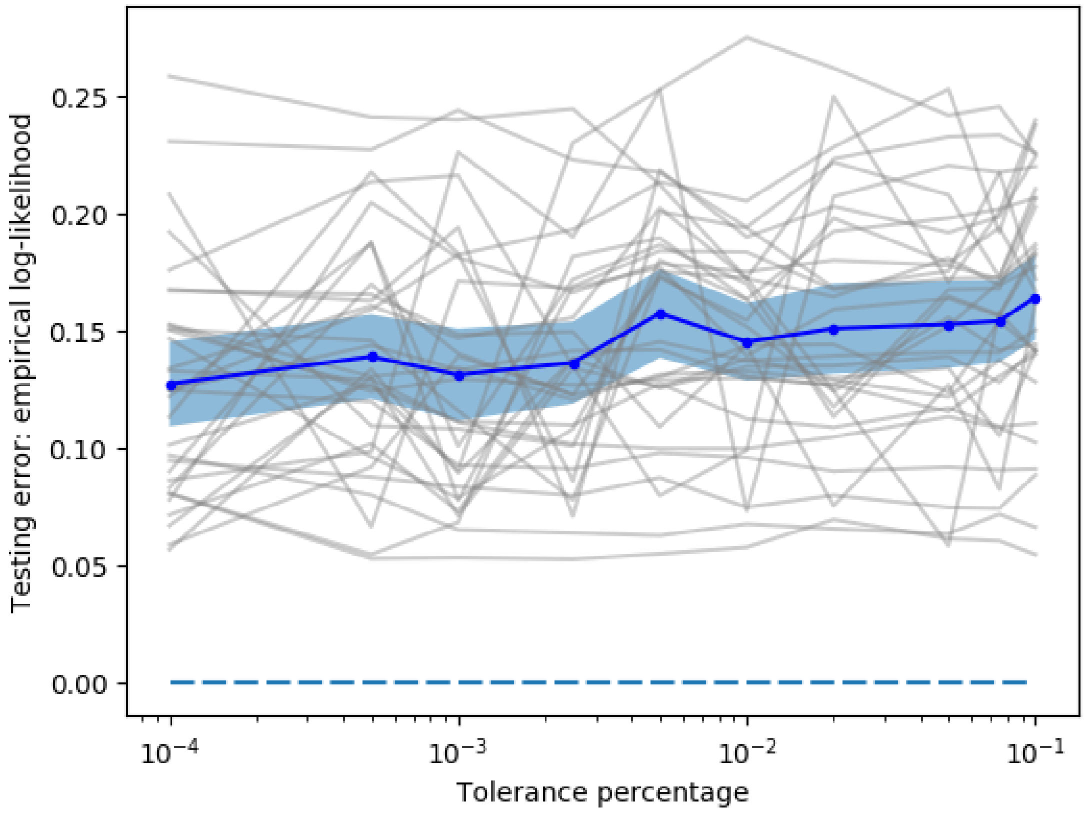


Figure 3.6. Divergence to generating distribution over testing dataset (of 26.4k samples), plotted against varying tolerance, using 1000 training samples and 2 components.

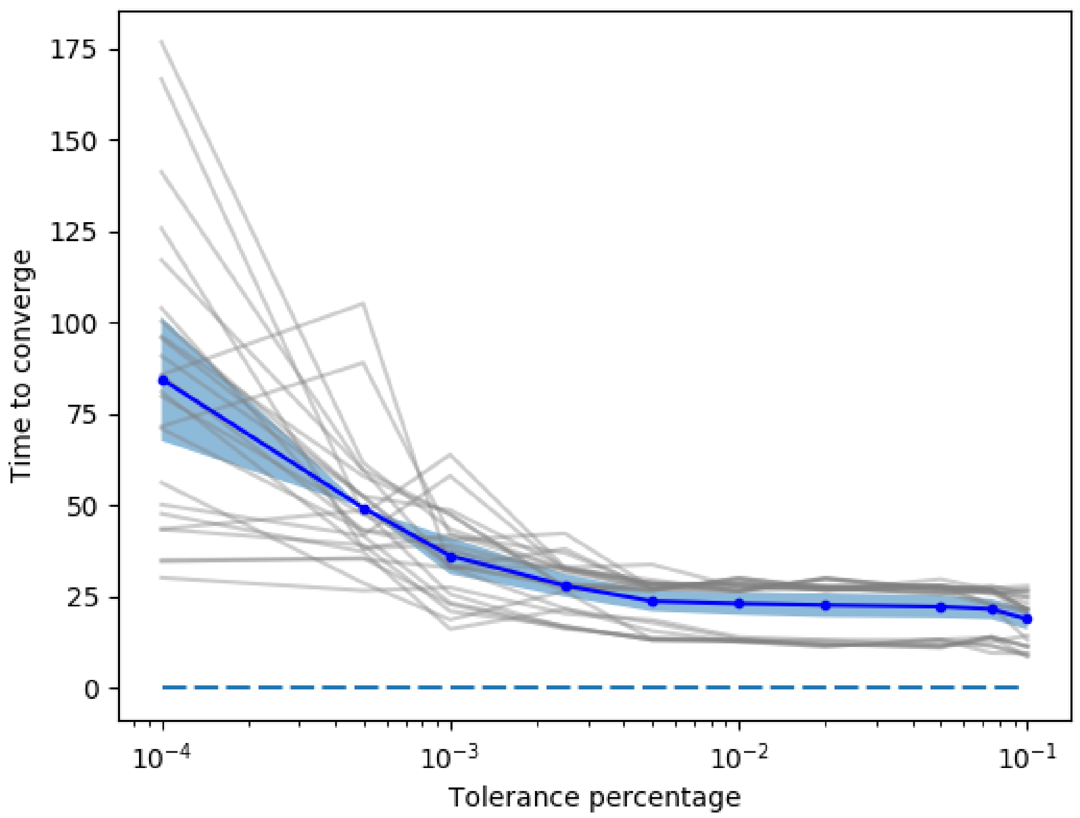


Figure 3.7. Learning time in seconds plotted against changing tolerance, using 1000 training samples and 2 components.

4. SUMMARY

4.1 Future work

There are several directions of future work that could build off of the modeling we have discussed. In this thesis, we have focused on the discrete alphabet setting, the framework should extend naturally to real-valued data, though there may be important implementation challenges. In this work, we focused on discrete valued data. In principle, the framework applies directly to continous valued data but there could be implementation challenges related to estimating mutual information (for the directed spanning tree sub-routine) or in other EM steps. Alternatively, continuous valued data could be discretized and our method could be applied, though that will increase loss. Another direction is for larger Markov orders. For simplicity we focused on Markov order 1 for both the hidden state and the observed data. In principle, our method can be extended in a straightforward manner for any Markov order. However, there will be a combinatorial explosion in parameters, so it would be impractical for large Markov orders. Lastly, one possible direction of future work is to present theoretical convergence guarantees for time-dependent (and otherwise) mixture models, and then hidden Markov state models with time-dependent observables. An analysis of the global convergence properties of the EM algorithm is provided for i.i.d data that fall out of mixtures of Gaussians [34]. Another more general characterization of the basin of convergence in finite-sample and population cases is explored in [35].

REFERENCES

REFERENCES

[1] Nan-Jung Hsu, Hung-Lin Hung, and Ya-Mei Chang. Subset selection for vector autoregressive processes using lasso. *Computational Statistics & Data Analysis*, 52(7):3645–3657, 2008.

[2] Ali Shojaie and George Michailidis. Discovering graphical granger causality using the truncating lasso penalty. *Bioinformatics*, 26(18):i517–i523, 2010.

[3] Yuval Nardi and Alessandro Rinaldo. Autoregressive process modeling via the lasso procedure. *Journal of Multivariate Analysis*, 102(3):528–549, 2011.

[4] Andrew Bolstad, Barry D Van Veen, and Robert Nowak. Causal network inference via group sparse regularization. *IEEE transactions on signal processing*, 59(6):2628–2641, 2011.

[5] Alexander Jung, Gabor Hannak, and Norbert Goertz. Graphical lasso based model selection for time series. *IEEE Signal Processing Letters*, 22(10):1781–1785, 2015.

[6] Christopher J. Quinn, Negar Kiyavash, and Todd P. Coleman. Efficient Methods to Compute Optimal Tree Approximations of Directed Information Graphs. *IEEE Transactions on Signal Processing*, 61(12):3173–3182, June 2013.

[7] Marina Meila and Michael I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1(Oct):1–48, 2000.

[8] Sergey Kirshner, Padhraic Smyth, and Andrew W. Robertson. Conditional Chow-Liu tree structures for modeling discrete-valued vector time series. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 317–324. AUAI Press, 2004.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[10] Jason K. Johnson, Diane Oyen, Michael Chertkov, and Praneeth Netrapalli. Learning planar Ising models. *The Journal of Machine Learning Research*, 17(1):7539–7564, 2016.

[11] Kevin P. Murphy. *Machine learning: a probabilistic perspective.* Cambridge, MA, 2012.

[12] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, May 1968.

[13] Leiserson Cormen and CE Leiserson andR. Rivest. *Introduction to algorithms*, 2, 1990.

[14] Jack Edmonds. Optimum branchings. *Mathematics and the Decision Sciences, Part*, 1(335-345):26, 1968.

[15] Andrey Y. Lokhov, Marc Vuffray, Sidhant Misra, and Michael Chertkov. Optimal structure and parameter learning of Ising models. *Science Advances*, 4(3):e1700791, March 2018.

[16] Guy Bresler, Elchanan Mossel, and Allan Sly. Reconstruction of Markov Random Fields from Samples: Some Easy Observations and Algorithms. *arXiv:0712.1402 [cs]*, December 2007. arXiv: 0712.1402.

[17] Ali Jalali, Christopher C. Johnson, and Pradeep K. Ravikumar. On learning discrete graphical models using greedy methods. In *Advances in Neural Information Processing Systems*, pages 1935–1943, 2011.

[18] Animashree Anandkumar, Vincent Y. F. Tan, Furong Huang, and Alan S. Willsky. High-dimensional structure estimation in Ising models: Local separation criterion. *The Annals of Statistics*, 40(3):1346–1375, June 2012.

[19] Nicolai Meinshausen and Peter Bhlmann. High-dimensional graphs and variable selection with the Lasso. *The Annals of Statistics*, 34(3):1436–1462, June 2006.

[20] Anima Anandkumar, Daniel J. Hsu, Furong Huang, and Sham M. Kakade. Learning mixtures of tree graphical models. In *Advances in Neural Information Processing Systems*, pages 1052–1060, 2012.

[21] H. Cao, V. Y. F. Tan, and J. Z. F. Pang. A Parsimonious Mixture of Gaussian Trees Model for Oversampling in Imbalanced and Multimodal Time-Series Classification. *IEEE Transactions on Neural Networks and Learning Systems*, 25(12):2226–2239, December 2014.

[22] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[23] Y. Ephraim, D. Malah, and B. Juang. On the application of hidden Markov models for enhancing noisy speech. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(12):1846–1856, December 1989.

[24] Miroslav Krn. Recursive estimation of high-order Markov chains: Approximation by finite mixtures. *Information Sciences*, 326:188–201, January 2016.

[25] Rishi Gupta, Ravi Kumar, and Sergei Vassilvitskii. On mixtures of Markov chains. In *Advances in neural information processing systems*, pages 3441–3449, 2016.

[26] D. Luo, H. Xu, Y. Zhen, B. Dilkina, H. Zha, X. Yang, and W. Zhang. Learning Mixtures of Markov Chains from Aggregate Data with Structural Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1518–1531, June 2016.

[27] Y. CHU. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400, 1965.

[28] R. E. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, March 1977.

[29] Harold N. Gabow, Zvi Galil, Thomas Spencer, and Robert E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, June 1986.

[30] James Massey. Causality, feedback and directed information. In *Proc. Int. Symp. Inf. Theory Applic.(ISITA-90)*, pages 303–305. Citeseer, 1990.

[31] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[32] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[33] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171, February 1970.

[34] Ji Xu, Daniel Hsu, and Arian Maleki. Global analysis of Expectation Maximization for mixtures of two Gaussians. *arXiv:1608.07630 [cs, math, stat]*, August 2016. arXiv: 1608.07630.

[35] Sivaraman Balakrishnan, Martin J. Wainwright, and Bin Yu. Statistical guarantees for the EM algorithm: From population to sample-based analysis. *The Annals of Statistics*, 45(1):77–120, 2017.