1-1-2016

Sequential Learning and Variable Length Markov Chains

Joshua Magarick  
*University of Pennsylvania, jmagarick@gmail.com*

Follow this and additional works at: [http://repository.upenn.edu/edissertations](http://repository.upenn.edu/edissertations)

*Part of the Statistics and Probability Commons*

**Recommended Citation**

[http://repository.upenn.edu/edissertations/1870](http://repository.upenn.edu/edissertations/1870)

This paper is posted at ScholarlyCommons. [http://repository.upenn.edu/edissertations/1870](http://repository.upenn.edu/edissertations/1870)

For more information, please contact libraryrepository@pobox.upenn.edu.
Sequential Learning and Variable Length Markov Chains

Abstract
Sequential Learning is a framework that was created for statistical learning problems where \( (Y_t) \), the sequence of states is dependent. More specifically, when it has a dependence structure that can be represented as a first order Markov chain. It works by first taking nonsequential probability estimates \( P(Y_t | X_t) \) and then modifying these with the sequential part to produce \( P(Y_t | X_{1:T}) \). However, not all sequential models on a discrete space admit such a representation, at least not easily. As such, our first task is to extend Variable Length Markov Chains (VLMCs), which belie their name and are not Markovian, to be used in the sequential learning framework. This extension greatly broadens the scope of sequential learning as using VLMCs permits sequential learning with far fewer assumptions about the underlying dependence of states. After developing the VLMC extension we provide an overview of sequential learning in general and investigate the probability estimates it produces both theoretically and with a simulation study to assess model performance as a function of the complexity of the underlying sequential model and the quality of the initial probability estimates. Next, we apply VLMC sequential learning to the original dataset and problem that inspired sequential learning --- that of scoring sleep in mice using video data. We find that VLMCs perform at the same level, tying and sometimes beating the previous best sequential method which required many assumptions about the sequence of sleep states and a much more rigid model of sequential dependence. Finally, we turn our attention to the problem of modifying predictors when marginal class probabilities are known. This is inspired by the fact that in sequential learning problems, the marginal class distribution can vary substantially from sample to sample in contrast to i.i.d. problems. We provide a general method of marginal probability reweighting, show it to be equivalent to several extant methods used on similar problems, and provide a proof that our method improves probability estimates under log loss. We conclude with simulations assessing our method as a function of loss type and classifier used.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Statistics

First Advisor
Abraham J. Wyner

Subject Categories
Statistics and Probability

This dissertation is available at ScholarlyCommons: http://repository.upenn.edu/edissertations/1870
ACKNOWLEDGEMENT

It is easy to thank people in words, but much harder to do so with actions. Many people have helped me so far and I’m sure many more will. If I tried to enumerate them all here, I would likely forget some. And if I could list them all here, what a long, long, list it would be. To me, it is far more meaningful to make sure they all know personally, and to show through my actions the magnitude of my gratitude. I hope I can do at least as much for others as others have done for me.
ABSTRACT

SEQUENTIAL LEARNING AND VARIABLE LENGTH MARKOV CHAINS

Joshua M. Magarick

Abraham J. Wyner

Sequential Learning is a framework that was created for statistical learning problems where \( (Y_t) \), the sequence of states is dependent. More specifically, when it has a dependence structure that can be represented as a first order Markov chain. It works by first taking nonsequential probability estimates \( P(Y_t | X_t) \) and then modifying these with the sequential part to produce \( P(Y_t | X_{1:T}) \). However, not all sequential models on a discrete space admit such a representation, at least not easily. As such, our first task is to extend Variable Length Markov Chains (VLMCs), which belie their name and are not Markovian, to be used in the sequential learning framework. This extension greatly broadens the scope of sequential learning as using VLMCs permits sequential learning with far fewer assumptions about the underlying dependence of states. After developing the VLMC extension we provide an overview of sequential learning in general and investigate the probability estimates it produces both theoretically and with a simulation study to assess model performance as a function of the complexity of the underlying sequential model and the quality of the initial probability estimates. Next, we apply VLMC sequential learning to the original dataset and problem that inspired sequential learning — that of scoring sleep in mice using video data. We find that VLMCs perform at the same level, tying and sometimes beating the previous best sequential method which required many assumptions about the sequence of sleep states and a much more rigid model of sequential dependence. Finally, we turn our attention to the problem of modifying predictors when marginal class probabilities are known. This is inspired by the fact that in sequential learning problems, the marginal class distribution can vary substantially from sample to sample in contrast to i.i.d. problems. We provide a general method of marginal probability reweighting, show it to be equivalent to several
extant methods used on similar problems, and provide a proof that our method improves probability estimates under log loss. We conclude with simulations assessing our method as a function of loss type and classifier used.
# TABLE OF CONTENTS

**ACKNOWLEDGEMENT** ................................................................. iii

**ABSTRACT** ........................................................................ iv

**LIST OF TABLES** ................................................................. viii

**LIST OF ILLUSTRATIONS** .................................................. x

**LIST OF ALGORITHMS** ....................................................... xi

**CHAPTER 1 : Introduction** .................................................... 1

**CHAPTER 2 : Markov Chains to Variable Length Markov Chains** .......... 5
  2.1 Fixed Order Markov Chains ............................................... 5
  2.2 Semi-Markov Models ..................................................... 6
  2.3 Variable Length Markov Chains ....................................... 10
  2.A Appendix: Detailed Context Tree Fitting ......................... 19

**CHAPTER 3 : Sequential Learning** ......................................... 21
  3.1 Hidden Markov Models .................................................. 21
  3.2 Discriminative HMMs .................................................... 30
  3.3 Results on Probability Estimation ................................... 32
  3.4 VLMC simulations ....................................................... 34

**CHAPTER 4 : Application: Mouse Sleep Data** ............................. 51
  4.1 The Data ..................................................................... 51
  4.2 Analysis ...................................................................... 57
  4.3 Results ...................................................................... 60

vi
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE 1</td>
<td>Sleep State by Time of Day</td>
<td>57</td>
</tr>
<tr>
<td>TABLE 2</td>
<td>Table of Bout Statistics</td>
<td>59</td>
</tr>
<tr>
<td>TABLE 3</td>
<td>Holdout Misclassification Rates</td>
<td>61</td>
</tr>
<tr>
<td>TABLE 4</td>
<td>Per-Mouse Misclassification Rates</td>
<td>62</td>
</tr>
<tr>
<td>TABLE 5</td>
<td>Holdout REM Performance by Method</td>
<td>73</td>
</tr>
<tr>
<td>TABLE 6</td>
<td>Per-Mouse REM Performance by Method</td>
<td>74</td>
</tr>
<tr>
<td>TABLE 7</td>
<td>Two State Holdout Misclassification</td>
<td>75</td>
</tr>
<tr>
<td>TABLE 8</td>
<td>Two State Per-Mouse Misclassification</td>
<td>75</td>
</tr>
<tr>
<td>TABLE 9</td>
<td>Fit Bout Statistics</td>
<td>76</td>
</tr>
<tr>
<td>TABLE 10</td>
<td>Length 1 Wake Bout Error Rates</td>
<td>77</td>
</tr>
<tr>
<td>TABLE 11</td>
<td>Entropy rates</td>
<td>84</td>
</tr>
</tbody>
</table>
LIST OF ILLUSTRATIONS

FIGURE 1: Sequentially dependent data. .............................................. 1
FIGURE 2: Example Context Tree ......................................................... 15
FIGURE 3: VLMC Expansion ............................................................... 17
FIGURE 4: Simulation tree transition probabilities ................................ 35
FIGURE 5: Simulation Trees ................................................................. 36
FIGURE 6: First Simulation Group Misclassification ............................ 39
FIGURE 7: First Simulation Group Log Loss ........................................ 40
FIGURE 8: First Simulation Group RMSE ............................................. 41
FIGURE 9: Second Simulation Group Misclassification ....................... 44
FIGURE 10: Second Simulation Group Log Loss ................................. 45
FIGURE 11: Second Simulation Group RMSE ...................................... 46
FIGURE 12: Third Simulation Group Misclassification ....................... 48
FIGURE 13: Third Simulation Group Log Loss ...................................... 49
FIGURE 14: Mouse Video Covariates by State .................................... 54
FIGURE 15: Mouse Video Covariates by State and Time .................... 55
FIGURE 16: Bout Duration by Mouse Violin Plot ............................... 58
FIGURE 17: REM ROC Curves .......................................................... 64
FIGURE 18: NREM ROC Curves ........................................................ 65
FIGURE 19: Wake ROC Curves .......................................................... 66
FIGURE 20: Mouse Pair Calibration Plot ........................................... 68
FIGURE 21: Holdout Calibration Plot .................................................. 69
FIGURE 22: Probability Estimate Histograms .................................... 71
FIGURE 23: Parameters for GMM simulations. ................................. 79
FIGURE 24: GMM Simulation Discrete Beta Performance .................... 82
LIST OF ALGORITHMS

ALGORITHM 1 : Grow Context Tree (high level) ........................................ 12
ALGORITHM 2 : Prune Context Tree ...................................................... 14
ALGORITHM 3 : Generate Minimal Markov Hull ..................................... 17
ALGORITHM 4 : Fit VLMC (Detailed) ..................................................... 20
ALGORITHM 5 : Forward Algorithm ...................................................... 25
ALGORITHM 6 : Backward Algorithm .................................................... 25
ALGORITHM 7 : Viterbi Algorithm ......................................................... 27
ALGORITHM 8 : Baum-Welch Algorithm ............................................... 28
ALGORITHM 9 : Add Context Trees ....................................................... 60
Chapter 1

Introduction

The objects of study in this dissertation are problems arising in the study of data with sequentially dependent labels. By this, we mean that in contrast to many machine learning problems where the covariate-label pairs, \((X_t, Y_t), t = 1, \ldots, T\), not independent. We presume a dependence structure in place determined by dependence among the \(Y\)s. Additionally, the models we consider treat \(X\) as being “emitted” from \(Y\), meaning that \(X_t\) is conditionally independent of all other \(X\)s given \(Y_t\). This structure is illustrated graphically in figure 1. As such, the goal is to estimate \(P(Y_t | X_{1:T})\) rather than \(P(Y_t | X_t)\) because without observing the labels, the entire sequence \(X_{1:T}\) carries information about each \(Y_t\).

Figure 1: Sequentially dependent data.
What we call the “Sequential Learning” framework was introduced in the context of modeling and scoring the sleep stages of mice using video data by McShane et al. [39]. In this case, we clearly do not believe that whether a mouse is awake, in REM sleep or in non-REM sleep is independent from time period to time period. It is also reasonable to assume that the observations $X_t$ at any given time depend only on the current state of the mouse.

However, while the original work provided a general method for any sequential process that can be embedded in a first order Markov chain, not all such models are easily representable in this way. The models used, Generalized and Transition Dependent Generalized Markov Models, are forms of Semi Markov Models that explicitly and parametrically model the holding times of each state. While these sequential models worked well with the data, they are parametric models and selecting both the families of holding time distributions and parameters used required careful inspection of the data and hand-selection of some parts of the model.

To this end, we want to apply Variable Length Markov Chains (VLMCs) to this problem. VLMCs belong to the class of tree-based sequential models commonly used in information theory and data compression and were introduced in the form we use by Bühlmann and Wyner [11]. Instead of modeling a sequence of states as a fixed order Markov chain or using parametric holding time distributions, VLMCs produce transition probabilities from a context function that depends on, as the name implies, a variable number of previous states. Since this history can be long when needed and short when not, VLMCs are able to parsimoniously model processes with a long memory. This approach generalizes both higher order and semi-Markov models and would allow sequential learning on more diverse data without the need to assume the labels come from a particular parametric process.

However, to do so requires that VLMCs be represented as a first order Markov model. And unfortunately, the standard fitting algorithm for VLMCs produces trees where this is not possible, so some modification is required. The naïve way of doing this would be to create a full $d^{th}$ order Markov chain where $d$ is the maximal context length. But this defeats the
purpose of a VLMC as we would end up with an exponentially large model that we sought to avoid. Instead, we would like a parsimonious model. Chapter 2 in addition to discussing VLMCs and how they generalize previous Markov models in sequential learning, provides a way to do this. We present a necessary and sufficient condition for a VLMC to have a first order embedding that has far fewer additional states than the number required by the simplistic method, and an algorithm to create it.

Following this, we turn our attention in chapter 3 to the problem of sequential learning. After reviewing previous work outlining the algorithm for use with any process representable as a first order Markov chain and any discriminative classifier we investigate the conditional class probability estimates produced by sequential learning. First, we prove three new results about the probability estimates sequential learning produces in different edge cases. While the assumptions are not realistic in practice, they serve to enhance our intuition and better understand phenomena that we observe in simulations. We conclude this chapter with a simulation study to better understand the relationship between sequential learning and the initial, nonsequential classifier used, finding that while sequential methods always help, a good nonsequential model can sometimes beat a bad one augmented with sequential learning.

Next, chapter 4 applies VLMC sequential learning to a real dataset consisting of the sleep states of mice and measured covariates taken from video recordings. As mentioned above, this is a good setting for the sequential learning paradigm, as there is signal in our covariates and the labels have clear and complex sequential dependence. Using VLMCs on this data performs similarly to, and often better than the previous best sequential method which used a parametric model on holding times and required far more human intervention to fit. This result is promising as sleep behavior differs across mice both on an individual and a strain level. VLMCs may allow us to model this behavior for sequential learning without spending time and effort trying to pick the right family of distributions to model holding times of different groups. Additionally, in this chapter we present new a method for fitting VLMCs
on multiple sequences, which allows us to build more accurate and robust models by fitting on multiple mice and thus dealing with some of the observed mouse-to-mouse variability.

Finally, chapter 5 addresses our largest contribution of improving probability estimates when the marginal distribution of labels is known. This question arose from a series of sequential learning simulations with surprising results that we begin the chapter by replicating. They ask what happens if we know one piece of a model in sequential learning — either the Markov part of the nonsequential conditional class probabilities, $P(Y | X)$. The initial results showed that, when measured by RMSE to the true $P(Y_t | X_{1:T})$, a model that estimated both parts seemed to do worse than one that knew one part or the other. As we show in an extension to these simulations, they did not correctly use $P(Y)$, the marginal class distribution and doing so resolves the problem. While the marginals matter little in standard statistical learning problems — they vary little from sample to sample — complex Markovian structures on the label yield empirical marginal distributions that can be quite different each time we simulate.

From this we investigate the general question of known marginals and provide a method for incorporating this information into any estimate of conditional class probability. Our problem is related to several others in the literature, and indeed, we are able to show that our method of marginal probability rescaling is equivalent to several others used to solve similar problems. In addition, we provide a proof that rescaled probability estimates are always better than non-rescaled ones and can even show the exact size of the improvement. Following this, we present simulations of the method using several classifiers and show that, at least in simulation, the method provides benefits under misclassification loss as well.
Chapter 2

Markov Chains to Variable Length Markov Chains

This chapter discusses Markov chains and some of their generalizations, which we will use in classification problems where the class labels are assumed to be dependent. The principal aims of this chapter are to outline the models, and show how generalizations such as the Variable Length Markov Chain can be represented as a standard Markov chain.

2.1. Fixed Order Markov Chains

2.1.1. First Order Markov Chains

**Definition 1.** If \((X_n)\) is a stationary sequence of random variables taking values in a discrete alphabet \(A\) we say that \((X_n)\) has the Markov property if

\[
P(X_n \mid X_{0:(n-1)}) = P(X_n \mid X_{n-1}).
\]

for all \(n\).

The Markov property is also called the memoryless property because the distribution over
outcomes for \(X_n\) depends only on \(X_{n-1}\), meaning we can “forget” anything observed before that. The Markov property lets us represent a Markov chain concisely in the form of a transition matrix \(A\) where \(A_{ij} = P(X_n = j \mid X_{n-1} = i)\).

### 2.1.2. Higher Order Markov Chains

As with first order Markov Chains, we can define a higher order Markov property where we are allowed to remember more than just the previous state.

**Definition 2.** A stationary sequence of random variables \((X_n)\) has the \(m\)th order Markov property if

\[
P(X_n \mid X_{0:(n-1)}) = P(X_n \mid X_{n-m}).
\]

for all \(n\).

**As First Order Markov Chains**

Any higher order Markov chain can be represented as a first order Markov chain by considering tuples of states. For instance, with a second order Markov chain, we might define the new process \(X'_t = (X_{t-1}, X_t)\), which now exists over \(\mathcal{A}^2\) - the set of sequences of length two of elements in \(\mathcal{A}\). We would now have a \(|\mathcal{A}|^2 \times |\mathcal{A}|^2\) transition matrix with \(|\mathcal{A}|^3\) nonzero entries, since each state of the new process \((X'_t)\) only has \(|\mathcal{A}|\) valid transitions.

Higher order Markov models allow for more complicated dependence relationships within the sequence, but can be problematic in practice. This comes from the fact that there are \(|\mathcal{A}|^m\) possible previous states the process can be in, meaning there are \(|\mathcal{A}|^{m+1}\) transitions of interest. This means that we would need exponentially more data to fit such models as as we raise the order of the chain.

### 2.2. Semi-Markov Models

Sometimes we would like to model a more complex process with longer memory, but either do not have enough data to estimate a Markov chain of the order we would like or believe
in some additional assumptions. One such class of assumptions is that the holding times of states — the number of time steps the chain spends in a state before exiting — follow some parametric distribution. To see why this is reasonable, consider the holding time for a first order Markov chain. We have

\[
P(X_{t+1} = a, \ldots X_{t+m-1} = a, X_{t+m} \neq a \mid X_t = a, X_{t-1} \neq a)
\]

\[
= P(X_{t+1} = a, \ldots X_{t+m-1} = a, X_{t+m} \neq a \mid X_t = a)
\]

\[
= P(X_{t+m} \neq a \mid X_{t+m-1} = a) \prod_{k=1}^{m} P(X_t + k = a \mid X_{t+k-1} = a)
\]

\[
= (P(X_2 = a \mid X_1 = a))^m (1 - P(X_2 = a \mid X_1 = a))
\]

which is a geometric distribution.

To resolve some issues in modeling higher order Markov process, one method has been to use Semi-Markov Models [48, 64], also known as Explicit Duration Markov Models [18], or Generalized Markov Models [38]. This class of models generalizes Markov chains by allowing states to have non-geometric holding times.

With a Markov chain, the transition matrix \( A \) governs the entirety of the process but a GMM adds the additional component of holding time distributions \( d_j \) for each \( j \in A \).

What this means is that when the process enters state \( a \) it also decides how long it will stay by sampling \( \delta \sim d_a \). So, if our observed sequence is \textbf{aaabbacccc}, then the sequence of states is \textbf{abac} and the sequence of times is \textbf{3 2 1 4}. This way of thinking about a process is considering it as a run length encoding.

 Thinking about GMMs like this is fine from a generative point of view, but if the holding time distributions are not geometric we need to make some modifications to represent this as if it were a first order model; i.e. with only a transition matrix. To start we will assume that the \( d_a \) have finite support, that is \( d_j(i) = 0 \) for \( i > M_j \) for some \( M_j \) for each \( j \).

With this in mind, we can create a first order Markov chain by considering “clock” and
state pairs $Z_t = (\delta_t, X_t)$ where $\delta_t$ is the value of the “clock” at time $t$ and $X_t$ is again the state of the process. That is, the sequence above would be written as

$$(3, a) (2, a) (1, a) (2, b) (1, b) (1, a) (3, a) (2, a) (1, a).$$

With this representation, we have $P(Z_{t+1} = (i-1,j) \mid Z_{t-1} = (i,j)) = 1$ for $i > 1$. When $i = 1$, $X_{t+1}$ must enter a different state and the countdown clock will reset.

When $i = 1$ we can compute

$$P(Z_{t+1} = (i,k) \mid Z_{t-1} = (1,j)) = P(\delta_{t+1} = i, X_{t+1} = k \mid Z_{t-1} = (1,j))$$

$$= P(\delta_{t+1} = i \mid X_{t+1} = k) P(X_{t+1} = k \mid Z_{t-1} = (1,j))$$

$$= d_k(i) A_{jk}$$

Note that in the second line we have used the fact that

$$P(\delta_{t+1} = i \mid Z_{t-1} = (1,j), X_{t+1} = k) = P(\delta_{t+1} = i \mid X_{t+1} = k)$$

because the holding time for the next state is independent of the previous state. As long as all the $d_k$ have finite support, we can see that we now have a first order Markov chain defined on $(Z_t)$. Of course, in principle $d_k$ could be an unbounded distribution, but then our chain would have infinitely many states. One way to handle this is to introduce a special state ‘+’ to the clock, calling the resulting models “GMM+”, as was done in McShane et al. [39] where we allow

$$P(Z_{t+1} = (+,j) \mid Z_{t-1} = (+,j)) = p_j > 0, \ P(Z_{t+1} = (M_j,j) \mid Z_{t-1} = (+,j)) = 1 - p_j$$

where $M_j < \infty$. In other words, each state $(+,j)$ has its own geometric holding time with
parameter $p_j$. The overall holding time distribution for state $j$ upon entry is

$$d_j(i) = q_j f_j(i) 1_{i \leq M_j} + (1 - q_j) g_j(i - M_j) 1_{M_j < i}$$

where $q_j$ is the probability of ending up in the “head” distribution as opposed to the geometric “tail” whose pmf is given by $g_j$. Also, $f_j$ is a proper PMF defined on $1, \ldots, M_j$; so this represents the probability of the clock taking on a certain value given that we end up in the head part of the distribution. To represent these as first order Markov chains, we simply note that

$$P(Z_{t+1} = (i,k) \mid Z_{t-1} = (1,j)) = q_k f_k(i) A_{jk} \quad \text{when } i \leq M_k$$

$$P(Z_{t+1} = (+,k) \mid Z_{t-1} = (1,j)) = (1 - q_k) A_{jk}$$

### Transition Dependent GMMs

We end this section with a brief discussion of Transition Dependent GMMs (TDGMM) [39]. While these are not the focus of this work, they are very similar to GMMs and we mention them for completeness and because they are compared against in later data analysis.

A TDGMM is a GMM where the next state holding time depends on which state the process is coming from as well as which it is going to. Therefore, instead of pairs, we now embed $X_t$ into $Z_t = (\delta_t, X_s, X_t)$ where again $\delta_t$ and $X_t$ are the state of the clock (possibly $+$) and original process respectively. Also $s = \max \{u : u < t, X_u \neq X_t\}$ which makes $X_s$ the previous state. Note now that if we are in state $(1,j,k)$, we must transition to a state of the form $(\delta, k, l)$ because the current state must, of course, become the previous state. Without
working through all of the details, the nonzero transitions are given by

\[
P(Z_{t+1} = (i, k, l) \mid Z_{t-1} = (1, j, k)) = q_{kl} f_{kl}(i) A_{jk} \quad \text{when } i \leq M_{kl}
\]

\[
P(Z_{t+1} = (+, k, l) \mid Z_{t-1} = (1, j, k)) = (1 - q_{kl}) A_{jk}
\]

\[
P(Z_{t+1} = (+, k, l) \mid Z_{t-1} = (+, k, l)) = p_{kl}
\]

\[
P(Z_{t+1} = (M_{kl}, k, l) \mid Z_{t-1} = (+, k, l)) = 1 - p_{kl}
\]

\[
P(Z_{t+1} = (i-1, k, l) \mid Z_{t-1} = (i, k, l)) = 1 \text{ if } 2 \leq i \leq M_{kl}
\]

with all of the holding time parameters originally indexed by one state simply being replaced by those indexed by two states.

2.3. Variable Length Markov Chains

As before, consider a stationary sequence of random variables \((X_n)\) taking values in a discrete alphabet \(A\), let \(A^n\) be sequences of length \(n\) and \(A^* = \bigcup_{n=0}^{\infty} A^n\) be sequences of arbitrary length. Now suppose that \(P(X_n \mid X_{n-1}, \ldots)\) does not depend on every previous state of the process, but that it does not depend on a fixed number of past states either. We describe such processes as Variable Length Markov Chains (VLMCs), since the dependence on previous states is Markov in nature, but not of a fixed order. Note that such models can theoretically be of infinite order \([12, 20, 15]\), although they can be approximated well by finite order models \([20]\) and as such are far less relevant to our discussion.

The term VLMC, introduced by Bühlmann and Wyner \([11]\) is one of many for a class of models designed to parsimoniously model data generated by processes that sometimes have a long memory and sometimes have a short memory. Similar models are known variously by names such as Probabilistic Suffix Automata (PSA) \([52]\), Context-Tree Weighting \([63]\), and Prediction by Partial Match \([14]\). While they differ in specifics such as implementation and how they are fit on data, the general idea of a finite, variable memory stochastic process represented by a tree is the same. For an overview of some such methods see Begleiter et al. \([5]\).
This is achieved by representing the state of the process with a *Context Function* and a *Context Tree*. The idea of representing a sequence of discrete states in such a tree goes back to the data compression algorithm of Ziv and Lempel [67] and the first relation of such trees to Markov Models by Rissanen [51]. Variable-order tree-based models have been used in many applications, including data compression [67], linguistics [25], and biology [42, 54].

**Definition 3.** A *Context Function* is a mapping $c : \mathcal{A}^* \to \mathcal{A}^*$ that has the property that $c(x)$ is a suffix, not necessarily proper, of $x$. In other words, for a (possibly infinite) sequence of observations, $c(x_{-\infty:0}) = x_{k:0}$ for some $k$.

**Definition 4.** If a context function has type $c : \mathcal{A}^* \to \bigcup_{i=0}^{k} \mathcal{A}^i$ for a fixed $k$, then we say it has order $k$. This means that the maximum length of the context of any string is $k$. These finite order context functions are at the base of VLMCs.

This definition, however, is too lenient for our purposes. We need

**Definition 5.** If $r$ is a prefix of $s$, denoted $r \preceq s$ implies that $c(r) \preceq c(s)$ we say that $c$ has the *prefix free* property and that it defines a *context tree*.

A context tree can be thought of as a representation of a context function where the value is determined by traversing a tree with edged labeled by the alphabet of the string and recording the path taken until a leaf is reached. Further, if we have a context tree $\tau$ defined by a context function $c$ we often use the notation $s \in \tau$ to mean that $s$ is a path to a leaf of $\tau$. This notation comes from the fact that context trees are often thought of in terms of their sets of leaves and allows us to abuse set theoretic notation to write $\tau \cup \{r\}$ to mean “add a leaf corresponding to the string $r$ to the tree $\tau$”.

One more piece of information is required to build probabilistic models out of context trees. Namely that each leaf in a context tree $\tau$ be associated with a probability distribution over $\mathcal{A}$. For this, we write $P(a|s) = P_s(a)$ to mean the probability of observing $a \in \mathcal{A}$ given that the context of everything we have seen is $s \in \mathcal{A}^*$. Sometimes we will also write $P_{\tau,s}(\cdot)$ or $P_{\tau}(\cdot|s)$, especially in the algorithms section to clarify that the distribution is one indexed by $\tau$. 

11
2.3.1. Fitting VLMCs

Here we describe the process of fitting a VLMC on a dataset. The method of finding contexts involves bottom up selection of contexts from the data as opposed to top down selection like some other methods \cite{52, 55, 61}. This allows the algorithm to find long contexts effectively when they exist without pre-specifying a maximal tree depth. In addition to describing the fitting at a high level, we also provide in the appendix pseudocode and a description of a clever dynamic programming algorithm used to fit context trees quickly that appears in the source of the R package VLMC \cite{35, 36} (and PyVLMC\footnote{See \url{https://github.com/magarick/pyvlmc} for information and source code.}) but to our knowledge has not been described anywhere else yet.

We begin with an overview of growing the tree, but first we will define a few terms. Denote the training sequence as $S$. Let $N(s)$ be the number of times the subsequence $s$ appears in $S$ and let $N(a|s)$ be the number of times the string (usually just one symbol) $a$ appears after $s$. Algorithm 1 describes this process. Algorithm 4 in the appendix describes it in more complete detail.

Algorithm 1 Grow Context Tree (high level)

Require: $S \in \mathcal{A}^*$, $k \geq 1$

\begin{verbatim}
\tau \leftarrow \emptyset
\text{for } s \in S \text{ do}
    \text{if } s \text{ appears in } S \text{ at least } k \text{ times then}
        \tau \leftarrow \tau \cup \{s\}
        \text{for } a \in \mathcal{A} \text{ do}
            P_{\tau,s}(a) \leftarrow N(a|s)/N(s)
        \text{end for}
    \text{end if}
\text{end for}
\text{return } \tau
\end{verbatim}

Next we describe the process of pruning a complete context tree. Here we consider the leaves of the tree and compute a measure of the distance between the estimated next state probability distribution of a leaf and that of its parent. If they are sufficiently different as determined by a pre-specified cutoff, then we remove the leaf from the tree. If the parent
becomes a leaf, we proceed upward. The pseudocode is in algorithm 2 and we describe the steps of the process in more detail below.

In order to do this, we have to pick a distance on probability distributions to use. Typically the distance used is

\[
D(p_{sr}, p_s) = \sum_{a \in A} N(a|sr) \log \left( \frac{\hat{p}_{sr}(a)}{\hat{p}_{s}(a)} \right) = \sum_{a \in A} \hat{p}_{sr}(a) \log \left( \frac{\hat{p}_{sr}(a)}{\hat{p}_{s}(a)} \right) N(sr) = N(sr) D_{KL}(p_{sr}||p_s)
\]

where \(D_{KL}\) is the Kullback-Leibler divergence between two probability distributions. This distance is then compared to a cutoff \(c\) and the child pruned if \(D(p_{sr}, p_s) < c\). Asymptotic considerations \([10, 11]\) give \(c\) as \((2 |A| + 4) \log(n)\) where \(n\) is the length of the training sequence; however in practice this produces trees far too small to be useful for finite training data because the cutoff is so strict that without a very large amount of data the resulting tree is a low order Markov model.

In practice we often use a fixed quantile of a \(\chi^2\) distribution — usually \(\alpha = 0.05\) — with \(|A|\) degrees of freedom because the preceding distance can be written as \( \log \left( \frac{\hat{p}_{sr}(S)}{\hat{p}_{\tau'(S)}} \right) \), where \(\tau\) is the unpruned tree and \(\tau'\) is the tree with just that one child removed. This is because the likelihood test is asymptotically \(\chi^2\) and the larger tree has \(|A| - 1\) more degrees of freedom (since each leaf is just a discrete distribution over \(A\)). This too, can become problematic as sequences become long and we fit trees that are too large. So, in practice, both the cutoff and the minimal number of observations function as tuning parameters to similar effect.

However, in our experiences with data, we find that while the trees may differ by adjusting these parameters, the change is seldom very large and predictions do not change much. Work such as Bühlmann \([10]\) appears more interested in the asymptotic behavior of the pruning cutoff \(c\) as it relates to approximation of a true underlying tree model. Further,
changing either of the two parameters will have the same effect of making the resulting tree larger or smaller. Given these experiences and the nature of the algorithm, for predictive tasks it seems most important to use sensible default values. Typically, we keep the default pruning parameter, \( c \), but make \( K \) a small multiple of the total number of states. This has the benefit of lower computational time — since we are not fitting the largest possible tree before pruning when we know low-count leaves almost always end up being pruned — while not generating undesirably small trees either.

Algorithm 2 Prune Context Tree

```plaintext
function PRUNE(\( \tau \), \( c \), \( D \))
    Children ← {}  
    for Each child \( \tau' \) of \( \tau \) that is not a leaf do  
        Children ← Children \( \cup \) \{PRUNE(\( \tau' \), \( c \), \( D \))\}
    end for  
    for All \( \tau' \) ∈ Children that are leaves do  
        if \( D(p_{\tau'}, p_{\tau}) < c \) then  
            Children ← Children \( \backslash \{\tau'\} \triangleright \) Prune when next state distributions are “close” the parent’s.
        end if
    end for
    \( \tau \).children ← Children  
    \triangleright \) New children are the pruned set.
    return \( \tau \)
end function
```

2.3.2. Markov Hulls of VLMCs

VLMCs are not Necessarily Markovian

The “Markov” in VLMC belies the fact that the standard fitting algorithm does not produce a proper Markov Chain of any order. The context trees produced will violate the memoryless property possessed by Markov Chains. The context we use at time \( t + 1 \) may be more than one symbol longer than what we used at time \( t \), thus requiring us to recall previous states that we had forgotten. More formally, we must always have \( |c(X_{1:(n+1)})| \leq |c(X_{1:n})| + 1 \) for a VLMC to have the Markov Property. To illustrate this, consider the following VLMC and sequence of states.
Example 1. Suppose our data are generated according to the tree in figure 2 and our alphabet is $\mathcal{A} = \{0, 1\}$. If we have just seen the symbols 0 0 then our context is unambiguously the leftmost leaf of the tree. If the next symbol we see is 1, again, we know what state we are in and have “forgotten” the previous two 0s. However, if we then see another 0 we are forced to look back beyond that symbol and the 1 we just saw. This violates the desired memoryless property.

The Minimal Markov Hull

Any VLMC has a trivial representation as a first order Markov chain. Simply take the longest context with length $L$ and extend all branches with all possible prefixes until they are length $L$ as well, propagating the probability distribution downward. Formally, let $\tau$ be the context tree, $L = \sup_s \{|s| : s \in \tau\}$ and define the extension

$$\tau_{\text{big}} = \{us : |us| = L, u \in \mathcal{A}^*\}$$

$$p_{us}(\cdot) = p_s(\cdot)$$

This is, of course, a terrible idea. $\tau_{\text{big}}$ now has $|\mathcal{A}|^L$ states, most of them redundant. It is the same as a standard $L^{th}$ order Markov model. While this is fine mathematically, it is a computational disaster. And the whole point of VLMCs was a small representation of a process! The question is, can we create a first order Markov chain from a VLMC without exponentially exploding the state space? Since you’re reading this, the answer is “yes”.

Figure 2: Context Tree for Example 1. Left branches represent seeing a 0, right branches a 1.
We now present a simplified version of a theorem from [37] which shows that VLMCs can be extended to have the memoryless property and that the number of leaves in the resulting context tree is no more than $O(|\tau|^2)$ (and in practice often less) where $|\tau|$ is the number of leaves in the original tree. By construction we see that this is the Minimal Markov Hull in that an extension with fewer states would not retain the memoryless property.

To show this, we begin with a few simple definitions and a proposition that tells provides necessary and sufficient conditions for a context tree to have the desired property.

**Definition 6.** For a context tree $\tau$, we denote $L(\tau)$ as the set of leaves of $\tau$.

**Definition 7.** Let $x_1:n$ be a sequence of symbols with $x_i \in A$. We define $\text{Head}(x) = x_n$ and $\text{Tail}(x) = x_1:(n-1)$

**Proposition 1.** We say that a context tree $\tau$ is memoryless if and only if for all $s \in L(\tau)$, $\text{Tail}(s)$ is a suffix, not necessarily proper, of some context $r \in L(\tau)$.

**Proof.** First we show that memorylessness implies the suffix property. Suppose we have a sequence $x_{1:n} \in A^*$ and a context $s \in L(\tau)$ with $c(x_{1:n}) = s = x_{(n-|s|+1):n}$. If the tree is memoryless, then we know that $c(x_1:(n-1)) = a \text{Tail}(s)$ for $a \in A^* \cup \{\epsilon\}$. In words, either we have picked up exactly one symbol of memory from time $n-1$ to $n$ or have remembered one additional symbol and forgotten at least one symbol. If this were not the case, and the context at time $n-1$ ended in a proper suffix of $\text{Tail}(s)$ then we would be reaching farther into the past from $n-1$ to $n$ and violating the memoryless property.

Now suppose the tree lacks the memoryless property. So, for some sequence $x_{1:n} \in A^*$ we have $c(x_{1:n}) = r_2r_1x_n$ where $|r_2| > 1$ and $c(x_{1:(n-1)}) = r_1$. Now suppose there is some other sequence $qr_2r_1 \in L(\tau)$ which would imply the suffix property. However, the fact that $c(x_{1:(n-1)}) = r_1$ implies that $r_1$ is a leaf and so such a sequence cannot exist. Thus, lacking the memoryless property means we lack the suffix property as well.

The preceding proposition shows that if we take a full tree $\tau$, we can create a memoryless tree by adding all prefixes of all contexts in $\tau$. The all prefixes condition comes from the
fact that if we add a context, we must then recurse and add the tail of the added context if it is not in the tree already. Algorithm 3 describes the process of extending the space of contexts, giving new ones the appropriate next state probabilities.

**Algorithm 3** Generate the Minimal Markov Hull from a fit VLMC. Takes only a fit tree \( \tau \) as an argument.

```plaintext
function ExtendTree(\( \tau \))
    contexts \( \leftarrow \{ s \mid s \in \tau \} \)
    for \( s \in \text{contexts} \) do
        for \( \ell \in \{1, \ldots, |s|\} \) do
            if \( s_{1:\ell} \notin \tau \) then
                \( r \leftarrow \) shortest string s.t. \( r \in \tau \) and \( s_{1:\ell} \) ends with \( r \)
                for \( m \in \{\ell - |r|, \ldots, 1\} \) do
                    Add \( s_{m:\ell} \) to \( \tau \)
                    \( p_{s_{m:\ell}}(\cdot) = p_r(\cdot) \)
                end for
            end if
        end for
    end for
end function
```

Going back to \( \textbf{1} \) we can see the results of expanding our simple tree both using the minimal Markov hull and by naively adding in all the states. Even in this case, we get a pleasantly smaller tree than the naive method.

![Figure 3: Expanded Trees from 1. Red nodes are the minimal ones needed to add and cyan nodes are unnecessary.](image-url)
An important consequence of proposition [1] is that every VLMC now has a matrix representation, as any ordinary Markov chain would. For any state in the expanded tree and any symbol $a \in \mathcal{A}$ we know the next state unambiguously. Refer to example [1] to see that this is not always the case for unexpanded trees. Additionally, this means that the matrix representation of a VLMC is sparse, with each row having only $|\mathcal{A}|$ nonzero elements, making it potentially useful for problems with a very large state space.

### 2.3.3. Variable Length Markov Models and Generalized Markov Models

GMMs, TDGMMs and similar models are thought of in terms of the holding time distribution for each state and use this to generate their first order embedding. With VLMCs, it is not immediately apparent how to compute holding time distributions from context trees and thus how to represent a GMM as a VLMC. However, it turns out to be reasonably simple.

Let $\mathcal{A}$ be the alphabet the VLMC is defined over and pick some $a \in \mathcal{A}$. Let $a^k$ be a sequence of $k$ repeated $a$s and let $P(a^k)$ be the marginal probability of such a sequence according to the VLMC. This is trivial to compute from the context tree probabilities. The holding time for state $a$ assuming the process $X_t$ enters $a$ at time $T$ is $H_a = \min_{t>T} \{t : X_t \neq a, X_T = a, X_{T-1} \neq a\} - T$. We can then compute probabilities as follows:

$$P(H_a = k) = P(H_a \geq k) - P(H_a \geq k + 1)$$

$$= P\left(X_{2;k} = a^{k-1} \mid X_0 \neq a, X_1 = a\right) - P\left(X_{2;(k+1)} = a^k \mid X_0 \neq a, X_1 = a\right)$$

$$= \frac{P\left(X_0 \neq a, X_{1;k} = a^k\right) - P\left(X_0 \neq a, X_{1;(k+1)} = a^{k+1}\right)}{P\left(X_0 \neq a, X_1 = a\right)}$$

$$= \frac{P(a^k) - 2P(a^{k+1}) + P(a^{k+2})}{P(a) - P(aa)}$$

We could also express the last line as $\frac{P(a^{k-1} \mid a) - 2P(a^k \mid a) + P(a^{k+1} \mid a)}{1 - P(a \mid a)}$. Note that if we are
dealing with a first order Markov chain, then this reduces to $P(a | a)^{k-1} (1 - P(a | a))$ as we would expect. Additionally, note that, if $M$ is the length of the longest context consisting entirely of $a$'s, then once we have observed $M$ in a row, the additional holding time will be geometric as in the case of a first order Markov chain.

This means that to turn a GMM into a VLMC, we would convert the holding time probabilities in its head distribution into conditional next-state probabilities and plug them into the distributions in the nodes of our VLMC. This, of course, has the downside of not being a nice parametric model anymore. Note also that we could repeat this analysis for the holding time of state $a$ conditional on coming from some other state $b$. However, this would not quite give us a TDGMM, as eventually we would reach a leaf of the VLMC and forget which state we came from. Another way to see this is that, technically, a TDGMM with an infinite state duration distribution is an infinite memory process. It must always remember the state it came from no matter how long it has been in its current state. This could, in principle be easily remedied by defining TD-VLMCs over state tuples as is done for TDGMMs, but we have not tried this in practice.

The above highlights both an important advantage and a disadvantage of VLMCs. They can capture a large class of Generalized/Semi Markov Models well, even without prior assumptions on state holding times. However, we give up some model parsimony and interpretability in the form of a small number of model parameters and we need large amounts of data to estimate complex trees.

2.A. Appendix: Detailed Context Tree Fitting

We describe in detail a dynamic programming algorithm for fitting context trees in the “grow” step of the VLMC algorithm. The basic premise of the algorithm is that if we are fitting on a sequence $s = (s_1, \ldots, s_n)$ and $w = (w_1, \ldots, w_k)$ is a substring of $s$, then we do not need to search all of $s$ for instances of $w$ if we keep track of the locations of the substring $(w_2, \ldots, w_k)$. And, since the nodes of the context tree represent these substrings,
keeping track at each node of the indices of the head of the context as well as the depth
lets us build the tree quickly since we have to examine less and less of the initial data at
each step.

For example, if \( A = \{a, b\} \) and we have the data \texttt{aabababaaba} then the index set for the
context of just \( b \) is \( \{3, 4, 7, 9, 12\} \). To build the children of that node we split the index set
into \( \{4\} \) for \( bb \) and \( \{3, 4, 7, 9, 12\} \) (ignoring the fact that we usually wouldn’t grow a node
for a context observed once in practice). Now, at each step, knowing the index set and
depth, we can quickly compute the transition probabilities since the number of elements in
the index set is the number of times the context occurs, and the next states are just the
symbol following the context heads.

Algorithm 4 Detailed Fitting procedure for VLMC context trees.

**Require:** \( s \in A^n, \ k \geq 1 \)

**Initialize:** index = 1, \ldots, n – 1, \( w \) to the empty string, \( \tau \) to an empty tree, \( d \) to 0

**function** FitTree(index, \( w \), \( d \))

\[
m \leftarrow |\text{index}|
\]

\[
\text{for } a \in A \text{ do}
\]

\[
p_{T,w}(a) \leftarrow |\{i : i \in \text{index}, s_{i+1} = a\}| / m
index_a \leftarrow \{i : i \in \text{index}, s_{i-d} = a\}
\]

\[
\text{end for}
\]

\[
\text{for } a \in A \text{ do}
\]

\[
\text{if } |\text{index}_a| \geq k \text{ then}
\tau_{wa} \leftarrow \text{FitTree(index}_a, wa, d + 1)
\]

\[
\text{end if}
\]

\[
\text{end for}
\]

**return** \( \tau \)

**end function**
Chapter 3

Sequential Learning

What we call “Sequential Learning” can be thought of as a case of what is sometimes known as “Structured Prediction” [53]. In this paradigm, we have observations \((X_t, Y_t)\) and wish to predict the \(Y\) given \(X\), but unlike in the most common learning scenario, the pairs are not i.i.d and we would like to use the dependence in our predictions.

In the case of Sequential Learning, we assume that, considered alone, the class labels \(Y\) form a discrete-time discrete space time series. In other words, we should be able to place some kind of Markov structure on \((Y_t)\). Making this assumption, it is natural that we proceed with methods derived from the classic and widely used Hidden Markov Models.

3.1. Hidden Markov Models

The theory of Hidden Markov Models (HMM) dates back to Baum and Petrie [3] and Stratonovich [58] and assumes that the data \((X_t, Y_t)\) are generated such that \((Y_t)\) comes from a first order Markov chain and \(X_t\) is “emitted” from \(Y_t\), so \(X_t\) and \(X_s\) are conditionally independent given \(Y_t\) or \(Y_s\) for \(s \neq t\). So, the parameters of interest for an HMM are:

- \(A\): The transition matrix on \((Y_t)\)
• $\pi$: The initial state distribution $P(Y_1)$, often taken to the the stationary distribution of $A$

• $f(\cdot \mid a)$ for $a \in A$: The emission distributions conditional on each possible value of the $Y_t$. For arbitrary discrete distributions over the $X_t$ we can think of this as the probabilities for each value conditional on $Y_t$ and for a parametric distribution on $X_t | Y_t$ we can think of this in terms of the distribution parameters $\lambda_a$.

For convenience, we often refer to the whole set of HMM parameters as $\vartheta$. With the definition in mind, Rabiner [49], in a popular paper on the model, poses three questions we can ask about such systems:

1. Given a sequence of observations $X_{1:T}$ and HMM parameters $\vartheta$, how can we efficiently compute $P(X_{1:T} \mid \vartheta)$ — the likelihood of a given sequence of observations?

2. Given a sequence of observations $X_{1:T}$ and HMM parameters $\vartheta$, how can we efficiently find a state sequence $\hat{y}_{1:T}$ that is optimal in “some meaningful sense” as Rabiner puts it. Typically this is taken to mean maximizing $P(X_{1:T} \mid \hat{y}_{1:T}, \vartheta)$ — the sequence that optimal in a maximum likelihood sense — or finding, for each $t$, the $\hat{y}_t$ that maximizes $P(\hat{y}_t \mid X_{1:T}, \vartheta)$

3. Given a sequence of observations $X_{1:T}$ and an unknown $\vartheta$, how can we find $\hat{\vartheta}$ such that $P(X_{1:T} \mid \hat{\vartheta})$ is maximized?

As we will see shortly, the first two questions can fit into the supervised learning framework easily, whereas it is less clear cut whether the third one can. However, we will discuss all three here and use both supervised and unsupervised techniques in our data examples. In what follows, we will first discuss the solutions given Rabiner’s questions and then show how the first two fit with the ideas of supervised learning as well as how they can work with extensions of first order Markov chains, as was first explained in McShane et al. [39].
3.1.1. The Forward-Backward Algorithm

The solutions to the first question and the second part of the second lie in the Forward-Backward algorithm. While not the only way to answer the questions, it is efficient. For instance, if we simply enumerated every possible set of values $Y_{1:T}$ could take on, we would have our answers. However, this requires $O(N^T)$ time, where $N$ is the number of states. In contrast Forward-Backward requires $O(N^2T)$.

To understand how this is possible, consider the conditional likelihood $P(Y_t \mid X_{1:T}, \vartheta)$ and the fact that $X_{1:t}$ and $X_{(t+1):T}$ are conditionally independent given $Y_t$ because of the HMM assumptions. Therefore, we can write

$$
P(Y_t \mid X_{1:T}, \vartheta) \propto P(X_{1:t} \mid Y_t, \vartheta) P(Y_t \mid \vartheta)
$$

$$
= P(X_{1:t} \mid Y_t, \vartheta) P(Y_t \mid \vartheta) P(X_{(t+1):T} \mid Y_t, \vartheta)
$$

$$
= P(X_{1:t}, Y_t \mid \vartheta) P(X_{(t+1):T} \mid Y_t, \vartheta)
$$

$$
\propto P(Y_t \mid X_{1:t}, \vartheta) P(X_{(t+1):T} \mid Y_t, \vartheta) \quad (*)
$$

The first term on the last line are known as the forward probabilities since they are the probability of $Y_t$ conditioned on everything up to time $t$ and we are predicting forward from observations. The second term are the backward probabilities since they are the probability of the emission sequence conditional on looking back at $Y_t$. Note that for each time $t$ we the forward and backward probabilities can be expressed as length $N$ vectors which we denote $\alpha_t$ and $\beta_t$. To make this useful, we now have to show how to compute each of these terms efficiently.

The desired efficiency comes from the following recursions which allow $O(N^2)$ operations
per term. To see this, note that

\[
P(Y_{t+1} \mid X_{1:(t+1)}, \vartheta) \propto P(Y_{t+1}, X_{1:(t+1)} \mid \vartheta)
\]

\[
= \sum_i P(Y_{t+1}, Y_t = i, X_{1:t}, X_{t+1} \mid \vartheta)
\]

\[
= \sum_i P(Y_{t+1}, X_{t+1} \mid X_{1:t}, Y_t = i, \vartheta) P(X_{1:t}, Y_t = i \mid \vartheta)
\]

\[
\propto \sum_i P(X_{t+1} \mid Y_{t+1}, X_{1:t}, Y_t = i, \vartheta) P(Y_{t+1} \mid X_{1:t}, Y_t = i, \vartheta) (\alpha_t)_i
\]

\[
= \sum_i P(X_{t+1} \mid Y_{t+1}, \vartheta) P(Y_{t+1} \mid Y_t = i, \vartheta) (\alpha_t)_i
\]

where the last line follows from the conditional independence in the HMM. Note that we can write this as \((\alpha_{t+1})_j = f(X_{t+1} | j) \sum_i A_{ij} (\alpha_t)_i \) and we have a recurrence (expressed more concisely in matrix form in algorithm 5). Also, notice that we need somewhere to start the algorithm, so we use \(\pi\), the initial state distribution of the Markov chain so

\[
P(Y_1 = j \mid X_1, \vartheta) = f(X_1 | j) \pi_j.
\]

Finally, notice that \(\sum_i (\alpha_T)_i\) gives the answer to Rabiner’s first question.

Next we compute the backward recursion:

\[
P(X_{t:T} \mid Y_{t-1}, \vartheta) = \sum_i P(X_{t:T}, Y_t = i \mid Y_{t-1}, \vartheta)
\]

\[
= \sum_i P(X_{t:T} \mid Y_{t-1}, Y_t = i, \vartheta) P(Y_t = i \mid Y_{t-1}, \vartheta)
\]

\[
= \sum_i P(X_{t:T} \mid Y_t = i, \vartheta) P(Y_t = i \mid Y_{t-1}, \vartheta)
\]

\[
= \sum_i P(X_{(t+1):T} \mid Y_t = i, \vartheta) P(X_t \mid Y_t = i, \vartheta) P(Y_t = i \mid Y_{t-1}, \vartheta)
\]

Since \(P(X_T \mid Y_{T-1}, \vartheta) = P(X_T \mid Y_T = i, \vartheta) P(Y_T = i \mid Y_{T-1}, \vartheta)\) we set \(\beta_T = 1\), the vector of all ones.

The forward and backward algorithms are formally written out in algorithms 5 and 6. Note also that because of all the “proportional to” terms, we rescale at every iteration to ensure
the terms are actually probabilities at each step. Finally equation \( P(Y_t = i \mid X_{1:T}, \vartheta) = \frac{(\alpha_t)_i(\beta_t)_i}{\alpha_t^T \beta_t^T} \) tells us that

Algorithm 5 Forward Algorithm

**Require:** \( F(t) \) are \( N \times N \) matrices with \( F_{ii}(t) = f(X_t \mid Y_t = i) \) and 0 otherwise, \( \pi \) is a length \( N \) vector of prior probabilities \( P(Y_1) \), and \( A \) is a Markov transition matrix.

\[
\begin{align*}
\alpha_1^T & \leftarrow \frac{\pi F(1)}{\|\pi F(1)\|_1} \\
\text{for } t & \in 2, \ldots, T \text{ do} \\
\alpha_t^T & \leftarrow \frac{\alpha_{t-1}^T A F(t)}{\|\alpha_{t-1}^T A F(t)\|_1} \\
\text{end for}
\end{align*}
\]

Algorithm 6 Backward Algorithm

**Require:** \( F(t) \) are \( N \times N \) matrices with \( F_{ii}(t) = f(X_t \mid Y_t = i) \) and 0 otherwise.

\[
\begin{align*}
\beta_T & \leftarrow \frac{1}{N} \\
\text{for } t & \in T - 1, \ldots, 1 \text{ do} \\
\beta_t & \leftarrow \frac{A \beta_{t+1} F(t+1)}{\|A \beta_{t+1} F(t+1)\|_1} \\
\text{end for}
\end{align*}
\]

3.1.2. The Viterbi Algorithm

Now we discuss the solution to the first part of the Rabiner’s second question. We want a sequence \( \hat{y}_{1:T} \) where \( \hat{y}_{1:T} = \arg\max_{Y_{1:T}} P(Y_{1:T} \mid X_{1:T}, \vartheta) \) which we note is equivalent to maximizing over the joint likelihood \( P(Y_{1:T}, X_{1:T} \mid \vartheta) \). The Viterbi algorithm [60, 23] produces a sequence of states that is optimal in this sense and, like the Forward-Backward algorithm, has a running time of \( O(N^2T) \) as opposed to the \( O(N^T) \) procedure that would be searching through every possible sequence. To achieve this, notice that if we were given a sequence of length \( T - 1 \) and its likelihood and we want to compute the most likely next state and the likelihood of this sequence we could write

\[
\max_{Y_T} P(Y_{1:(T-1)}, Y_T, X_T \mid \vartheta) = \max_{Y_T} P(Y_T, X_T \mid Y_{T-1}, \vartheta) \cdot P(Y_{1:(T-1)}, X_{1:(T-1)} \mid \vartheta).
\]
Notice that the second term on the right has no $Y_T$ in it, has the same form as our original objective function, and the first term is equivalent to $f(X_T | Y_T)P(Y_T | Y_{T-1}, \vartheta)$. Now we need to see how this recursion gets us what we want, since the most likely extension of a sequence does not guarantee that the whole sequence is now the most likely. However, if for every time $t$ we keep track of the most likely sequences such that $Y_t = i$, that is conditional on the sequence ending in state $i$, we will get what we want. To show this, begin by defining $\delta_t(i)$ as the likelihood of the most likely sequence up to time $t$ ending in state $i$. Then, we have

$$\delta_{t+1}(j) = \max_i f(X_{t+1} | j) A_{ij} \delta_t(i)$$

because the most likely sequence of length $t + 1$ ending in state $j$ must come from one of the previous most likely sequences since the change in likelihood from extending the sequence by 1 only depends on the transition matrix $A$, the previous state, and the emission distribution.

With this recurrence, we are almost done, but $\delta_t$ only takes care of the value we are maximizing, not the argument, so we also keep track of $\psi_{t+1}(j) = \arg\max_i A_{ij} \delta_t(i)$, which is the state from which we came at time $t$ to get to state $j$ in the most likely sequence of length $t + 1$ ending in state $j$. The final step to get the most likely path $\hat{y}_{1:T}$ is called backtracking; if we know that $\hat{y}_{1:t}$ ends with $\hat{y}_t = i$, then $\psi_{t-1}(i)$ will contain the state that got us there, $\hat{y}_{t-1}$, along the most likely path. This tells us that the most likely sequence of length $t - 1$ ends in $\hat{y}_{t-1}$, so continuing backward gets us the full sequence. Algorithm 7 gives the whole process in pseudocode.

3.1.3. The Baum-Welch Algorithm

The final HMM algorithm we consider is the solution given to Rabiner’s third question. Given observations $X_{1:T}$, we want HMM parameters $\lambda$ that maximize $P(X_{1:T} | \lambda)$. The Baum-Welch algorithm\[2, 4\] provides an iterative solution in the vein of the Expectation-Maximization (EM) algorithm\[17\], which works by starting with random HMM parameters $\vartheta^{(0)}$, using these in the Baum-Welch algorithm, and then using those estimated state prob-
Algorithm 7 Viterbi Algorithm

Initialize: $\delta_1(i) \leftarrow \max_j f(X_1|j)A_{ij}\pi_j$ for all $i$. ▷ Most likely sequence of length 0 is $\pi$.

Initialize: $\psi_1(i) = 0$. ▷ We’re not predicting where we came from at time 1.

for $t \in 1, \ldots, T - 1$ do
    for $i \in 1, \ldots, N$ do
        $\delta_{t+1}(j) = \max_i f(X_{t+1}|j)A_{ij}\delta_t(i)$
        $\psi_{t+1}(j) = \arg\max_i A_{ij}\delta_t(i)$
    end for
    $\hat{y}_T \leftarrow \arg\max_j \delta_T(j)$
    for $t \in T - 1, \ldots, 1$ do
        $\hat{y}_t = \psi_{t+1}(\hat{y}_{t+1})$
    end for
end for

abilities to update the estimate of $\vartheta$.

To do this, suppose we are given forward probabilities $\alpha_{1:T}$, backward, probabilities $\beta_{1:T}$. Then, as with the Forward-Backward algorithm, we compute $(\gamma_t)_i = \frac{(\alpha_t)_i(\beta_t)_j}{\alpha_T^j \beta_T^j}$, from which we can first estimate $\pi$, the initial state distribution as $\hat{\pi} = \gamma_1$. Next, we want to estimate $A$, the transition matrix, which can be done by computing

$$\hat{A}_{ij} = \frac{\sum_{t=1}^{T-1} P(Y_t = i, Y_{t+1} = j \mid X_{1:T}, \vartheta)}{\sum_{t=1}^{T-1} (\gamma_t)_i} \quad (3.1)$$

since this works out to be the average proportion of time we expect $Y$ to go from state $i$ to state $j$. To express this in terms of quantities we have already, write (suppressing $\vartheta$)

$$P(Y_t = i, Y_{t+1} = j, X_{1:T}) = P(X_{1:T} \mid Y_t = i, Y_{t+1} = j) P(Y_t = i, Y_{t+1} = j)$$

$$= P(X_{1:t} \mid Y_t = i) P(X_{(t+1):T} \mid Y_{y+1} = j) P(Y_t = i, Y_{t+1} = j)$$

$$= P(X_{1:t} \mid Y_t = i) P(X_{(t+1):T} \mid Y_{y+1} = j) P(Y_t = i) A_{ij}$$

$$= (\alpha_t)_i P(X_{(t+2):T} \mid Y_{y+1} = j) P(X_{t+1} \mid Y_{y+1} = j) A_{ij}$$

$$= (\alpha_t)_i (\beta_{t+1})_j f(X_{t+1}^j) A_{ij}$$

which is proportional to the quantity we want. Thus, just normalize this and we are done
because the desired quantity is proportional to what we just computed. Finally, we need to compute the emission distributions. The original Baum-Welch algorithm assumed that the $X_t$ were discrete scalar quantities and so would compute

$$
\hat{P}(X = k | Y = i) = \frac{\sum_{t: x_t = k} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)}
$$

(3.2)

where the numerator is the expected number of times we observe a $k$ and are in state $i$ and the denominator is the expected time spent in state $i$. In practice, however, we are not restricted to this, since if we assume $f(x|y)$ comes from some parametric distribution with parameter(s) $\lambda$, we can optimize with an “M” step from the EM algorithm, since the joint likelihood is

$$
P(X_{1:T}, Y_{1:T} | \vartheta) = P(X_{1:T} | Y_{1:T}, \vartheta) P(Y_{1:T} | \vartheta) = P(Y_{1:T} | \vartheta) \prod_t P(X_t | Y_t, \vartheta)
$$

and we have already taken care of the term outside of the final product. With this, we have what we need to write out algorithm 8

---

### Algorithm 8 Baum-Welch Algorithm

**Initialize:** $A^{(0)}$ as a random transition matrix. $f^{(0)}$ as random emission parameter(s).

$i \leftarrow 0$

**repeat**

- Compute $\gamma_t(i) \leftarrow \hat{P}(Y_t = i | X_{1:T}, \vartheta^{(i)})$ for all $i, T$, with Forward-Backward.
- $\pi^{(i)} \leftarrow \gamma_t(1)$
- Compute $A^{(i)}$ using equation 3.1
- Either use equation 3.2 or optimize for $\lambda$ to get $f^{(i)}$.

$i \leftarrow i + 1$

**until** Convergence

---

One important thing to note is that while the algorithm has been shown to converge and performs well in practice, it does not have to converge to the globally optimal parameters — much like EM. With this in mind, Juang and Rabiner [32] proposed the Segmental K-Means algorithm, also known as Viterbi Training. The difference between Viterbi Training and Baum-Welch is that VT computes the Viterbi-optimal path at each iteration and uses
those states to update the transition and emission probabilities instead of using the Forward-Backward probabilities. In practice, its chief advantage is in computational time, although it often underperforms the Baum-Welch algorithm.

3.1.4. Higher order HMMs

Any of the preceding methods can work with higher order HMMs and extensions of Markov chains such as those described in the preceding chapter. This holds as long as the process generating $Y_{1:T}$ can be described by a first order transition matrix over some set. In this case, we can use the preceding algorithms to compute paths or class probabilities over the expanded space and then return to the original one by either taking the corresponding original state or computing $P(Y_t | X_{1:T}, \theta) = \sum_{Y_t' \simeq Y_t} P(Y_t' | X_{1:T}, \theta)$ where $Y_t'$ is an expanded state and $Y_t' \simeq Y_t$ means $Y_t'$ has $Y_t$ as its observed state. Going forward we refer to the original set of states as $\mathcal{A}$ and the embedded set of states as $\mathcal{A}'$.

In the case of HMMs with specified holding time distributions, such as the GMMs or TDGMMs described previously, recall that the transition matrix is sparse. Most entries are 0 or 1 since states of the form $(i, k)$ ($(i, k, l)$ for Transition-Dependent models) where $i > 1$ just transition to $(i - 1, k)$. For example take a GMM over an alphabet where $|\mathcal{A}| = K$ and, for simplicity all symbols have maximum holding time $M$. The transition matrix over embedded states is $KM \times KM$ and has $K(K - 1)M + K(M - 1)$ nonzero entries which implies the HMM algorithms have running time $O(K^2MT)$. Such embeddings have used in McShane et al. [39] and as far back as Ramesh and Wilpon [50]. Further, Yu and Kobayashi [65] propose a faster algorithm in this special case, although it has not been extended to our more complex models and doing so is beyond our scope.

For VLMCs, the size of the embedding is determined by the size of the context tree. Since the states of interest are the tree’s leaves we have a $|\tau| \times |\tau|$ transition matrix with $K |\tau|$ nonzero entries. The drawback is that context trees can become very large for complex processes. However, we have not had an issue with speed in practice and the ability to fit
processes largely unconstrained by assumptions so far outweighs the additional complexity.

3.2. Discriminative HMMs

In its original form, the HMM is a fully generative model, meaning that for any set of values $X_{1:T}, Y_{1:T}$ we can compute the complete joint distribution of observations and states $P(X_{1:T}, Y_{1:T})$. This stands in contrast to a discriminative model which only seeks to estimate $Y$ conditional on $X$. An example of this would be Linear Discriminant Analysis versus Logistic Regression. Discriminative models are desirable when we are either not interested in the marginal distribution of the covariates or when such estimation would be intractable.

Also, note that what we are calling discriminative HMMs here are not true Hidden Markov Models. In order to apply discriminative methods, we require a training set with observed “hidden” states and hence assume that the labels observed are the only ones possible. In a truly hidden model, the number of states can be allowed to vary and are not always tied to any meaningful reality. It can often be though of, in fact, as a tuning parameter of the model, chosen by model selection procedures such as those that maximize a penalized likelihood [13, 44, 34, 57].

Fitting a discriminative HMM [39] thus operates in two phases. In the first, a procedure off-the-shelf or otherwise, is used to compute $\hat{P}(Y_t \mid X_t)$ and $\hat{P}(Y_t)$, the conditional and marginal class probabilities, not taking into account the sequential dependence of the labels $(Y_t)$. Separately, we fit a sequential model of our choice to $(Y_t)$.

Next, given the two pieces of our model, we want to generate estimates $\hat{P}(Y_t \mid x_{1:T})$ with a new unlabeled dataset. If we fit a first order Markov Chain to $(Y_t)$, or just happened to be given the true $P(Y_t \mid X_t)$ and $P(Y_t)$, it is simple to do. Notice that the Forward-Backward and Viterbi algorithms have $P(X_t \mid Y_t)$ in every update step and that $P(X_t \mid Y_t) = \frac{P(Y_t \mid X_t)P(X_t)}{P(Y_t)}$. Since $P(X_t)$ does not depend on $Y_t$, this term gets normalized away in the Forward-Backward algorithm and factors out of the maximization in the Viterbi algorithm. Therefore, anywhere we see a $P(X_t \mid Y_t)$ it can be replaced by $\frac{P(Y_t \mid X_t)}{P(Y_t)}$ or an
estimate thereof. Once we have done so, we can simply run the algorithms as before to get state probabilities or the most likely state sequence since the transition matrix requires no special treatment. Using anything other than a first order Markov Model requires a condition and an assumption:

1. The sequential model can be represented as a first order Markov Chain over some set of underlying states \((Y_t')\)

2. The conditional covariate distributions \(P(X_t | Y_t' = y_1')\) and \(P(X_t | Y_t' = y_2')\) are the same if both \(y_1'\) and \(y_2'\) are \(\approx y\) for some \(y\) in the original state space.

The second assumption requires some explanation. Our nonsequential classifier has by this point returned \(\hat{P}(Y_t | x_t)\) and \(\hat{P}(Y_t)\) but we have estimated a transition matrix \(\hat{A}'\) on the sequence \((Y_t')\). Because our algorithms require a first order Markov model, we have no choice but to first compute \(\hat{y}_{1:T}\) or \(\hat{P}(Y_t' | x_{1:T})\). But the true labels are in the set \(\mathcal{A}\), the values that appear in \((Y_t)\) and these are what the nonsequential classifier is trained on.

Fortunately, the assumption gets us around this because it implies that if \(y_1' \approx y\) and \(y_2' \approx y\), then the conditional class probabilities we assign these states are only a function of their marginal distribution and what the classifier gives for \(y\). Formally, and to see why this is relevant to our problem, by Bayes’ rule we have

\[
\frac{P(Y_t' = y_1' | X_t)}{P(Y_t' = y_1')} = \frac{P(Y_t' = y_2' | X_t)}{P(Y_t' = y_2')}. 
\]

Since all \(y'\) corresponding to the same \(y\) have equivalent ratios, it implies they are also equivalent to

\[
\frac{P(Y = y | X_t)}{P(Y = y)}. 
\]

This allows us to plug in \(\frac{P(Y = y | X_t)}{P(Y = y)}\) in place of the emission distributions in the Forward-Backward and Viterbi algorithms.\(^1\) The \(P(X_t)\) that comes out of applying Bayes’ rule is not relevant due to the scaling performed at each step.

With this, we can now compute \(\hat{P}(Y_t = y | x_{1:T})\) or \(\hat{y}_{1:T}\) by just plugging in our estimates of \(\hat{A}\) and \(\frac{P(Y = y | X_t)}{P(Y = y)}\) into the Forward-Backward and Viterbi algorithms respectively. Notice that if we knew the true process that generated \((Y_t)\) as well as the conditional and marginal class probabilities that this probability is indeed correct. However, as we note in the next

\(^1\)Notice that this doesn’t work for Baum-Welch because it needs to learn all of the model parameters.
section, this kind of smoothing pushes class probabilities toward zero and one. This implies that the method must be used with care, since this happens regardless of the accuracy of the sequential model or initial class probability estimates.

3.3. Results on Probability Estimation

We now turn our attention to some basic, but provable, properties of the discriminative HMM sequential learning algorithm as it relates to probability estimation. While the results are for extreme cases, they provide some intuition as to why the algorithm behaves as it does, having the tendency to push conditional class probabilities toward 0 and 1.

Let $F(t)$ by the diagonal matrix of scaled nonsequential conditional class probabilities at time $t$, so $F_{ii}^{(t)} = \hat{P}(Y_t = i|X_t)$ and $F_{ij}^{(t)} = 0$ for $i \neq j$.

Let $A$ be the Markov transition matrix and let $p_S$ be the stationary distribution across the states $Y$ can take on.

**Proposition 2.** Suppose that there is no Markov structure on the underlying states. That is, each observation is independent. And further assume that we set each row of $A$ to $p_S$, the stationary distribution of $Y$. Then, the estimated conditional class probabilities will be unchanged. That is $\hat{P}(Y_t|X_{1:T}) = \hat{P}(Y_t|X_t)$.

**Proof.** To see this, first note that for all $i, j$ we have $(AF(t))_{ij} = \hat{P}(Y_t = i|X_t)$. Then recall that the forward and backward probabilities can be written as $\alpha_t^T = \alpha_{t-1}^T A F(t)$ and $\beta_t = A F(t) \beta_{t+1}$. Assuming the $\alpha_t$ are normalized to sum to 1, this implies that

$$\alpha_t^T = [\hat{P}(Y_t = 1|X_t), \ldots, \hat{P}(Y_t = k|X_t)].$$

Further, since the backward algorithm starts with $\beta_T = 1$, this implies that $\beta_t = 1$ for all $t$, thus proving the proposition.

**Proposition 3.** Suppose that the base classifier returns $p_S$ for every observation. In other words, it knows the stationary distribution but any covariates provided are useless for pre-
dicting \( Y_t \). Further suppose that \( \pi \), the initial state distribution is also \( p_S \). Then, regardless of the Markov structure \( A \) has, if it has stationary distribution \( p_S \), the conditional class probabilities remain unchanged.

**Proof.** In this case we have \( F^{(t)}_{ii} = 1 \) because \( \hat{P}(Y_t = i|X_t) = \hat{P}(Y_t = i) \). So when we compute \( \alpha_2 \) in the first step of the forward algorithm, we get \( \alpha_2 = p_S^T A F^{(2)} = p_S^T A = p_S^T \). Inductively, we can see that \( \alpha_t = p_S \) for all \( t \). Next, in the backward algorithm we have \( \beta_{T-1} = A F^{(T-1)} 1 = A 1 = 1 \). Again, we can conclude that \( \beta_t = 1 \) for all \( t \). Thus, \( \hat{P}(Y_t|X_{1:T}) = \hat{P}(Y_t|X_t) \) for all \( t \). \( \square \)

Note that in the proceeding proposition, we have almost the same result even if not starting from the stationary distribution because as long as a unique stationary distribution exists, we converge to it at an exponential rate \([40]\). Thus, if our only information is in the form of an initial state distribution, there is little to gain.

While the preceding two propositions deal with how the algorithm behaves when we are lacking information, the next describes when we, in a sense, know too much. The intuition is that if we know for sure what one state is and there is no randomness in generating the sequence of states, then this provides enough information to know everything about the entire sequence.

**Proposition 4.** If the transition matrix \( A \) describes a deterministic process where, for each \( i \), there is a \( j \) so that \( A_{ij} = 1 \) and \( A_{ij'} = 0 \) for \( j \neq j' \) and if we have one time point \( s \) where there is an \( i \) such that \( \hat{P}(Y_s = i|X_s) = 1 \), then regardless of the predictions at other times, we will have \( \hat{P}(Y_t = i|X_{1:T}) \in \{0, 1\} \) for all \( i \) and \( t \).

**Proof.** Consider the forward probability moving from time \( s - 1 \) to \( s \) and suppose \( F^{(s)}_{ii} = 1 \) with all other entries being 0. Let \( j \) be the state uniquely transitioned to from state \( i \) and let \( e_j \) be the vector where the \( j^{th} \) entry is 1 and the rest are 0. We have

\[
\alpha^T_s = \alpha^T_{s-1} A F^{(s)} \propto e_j^T,
\]
which implies that $\alpha_{s+1} \propto e_{j'}$ where $j \to j'$. Continuing inductively, all forward probabilities beyond time $s$ must have the same form as well.

Next consider the backward probabilities. Let $i'$ be the state such that $i' \to i$. We have

$$\beta_{s-1} = AF^{(s)}\beta_s \propto e_{i'}.$$

Again, we see that inductively all $\beta_t$ where $t \leq s$ have this form. And so, $\hat{P}(Y_t = i|X_{1:T}) \in \{0, 1\}$ for all $t$ and all $i$.

Obviously this situation is not realistic in practice, but we have observed phenomena approaching this behavior when the label-generating process is closer to deterministic and the nonsequential Bayes error rate is also very low.

**Remark.** As we note from simulations and analyses on data, the class probability estimates using sequential data tend to be more toward 0 or 1 than nonsequential estimates. To understand why this is, recall that conditioning reduces the entropy of a random variable \[16\]. In this case, we can write $H(Y_t|X_{1:T}) \leq H(Y_t|X_t)$.

### 3.4. VLMC simulations

Here we demonstrate the performance of sequential learning using VLMCs with several sets of simulations, all over an alphabet of size 3, $\mathcal{A} = \{a, b, c\}$. In each set of simulations, we use variants on a particular sequential structure on the labels ($Y_t$) and generate the covariates from the same set of distributions in an attempt to focus on the effect different sequential structures have on performance. In each simulation, the total number of observations, $n$, was varied between 500, 2000, or 8000.

The first set consists of two synthetic VLMCs, as well as a first order Markov chain, the contexts and transition probabilities for one being given in figure [4]. The other is similar and varies in size and shape of the tree and used similar functions to generate next state probabilities. The full shape of each tree is also given in figure [5]. Full source code for the
simulations is provided online.

The second set of three simulations generates data from VLMCs fit on a real data source — the sleep states of mice discussed in chapter 4 — which generates far larger and more interesting trees than we could create by hand, containing dozens to hundreds of distinct states. The trees are of depth 12, 62 and 111 and have 67, 197, and 361 leaves respectively.

\[
P(a | b) = \frac{1}{3} \quad P(b | b) = \frac{1}{2} \quad P(c | b) = \frac{1}{6} \\
P(a | c) = \frac{1}{3} \quad P(b | c) = \frac{1}{6} \quad P(c | c) = \frac{1}{2} \\
P\left(a \mid a'\right) = \frac{1}{\sqrt{1+2}} \quad P\left(b \mid a'\right) = \left(1 - \frac{1}{\sqrt{1+2}}\right) / 2 \quad P\left(c \mid a'\right) = \left(1 - \frac{1}{\sqrt{1+2}}\right) / 2 \\
P\left(a \mid a'c\right) = \frac{9}{10} |\sin(i/2)| \quad P\left(b \mid a'c\right) = \frac{1}{4} \left(1 - \frac{9}{10} |\sin(i/2)|\right) \quad P\left(c \mid a'c\right) = \frac{3}{4} \left(1 - \frac{9}{10} |\sin(i/2)|\right) \\
P\left(a \mid a^{15}\right) = \frac{1}{2} \quad P\left(b \mid a^{15}\right) = \frac{1}{4} \quad P\left(c \mid a^{15}\right) = \frac{1}{4} 
\]

Figure 4: Transition probabilities for simulated tree 1. \(i\) ranges from 1 to 14

The third set of simulations use a process that generates states as follows: Define a fixed window size \(w\) (in our simulations we used, \(w = 10, 20, 40\)). With probability 0.95 the next state probabilities were given by \(P(Y_{t+1} = a \mid Y_{(t-w):t}) = \#\{y \in Y_{(t-w):t} \mid y = a\}/w\) and with probability 0.05 the next state is chosen uniformly from the three possible states. This allows the process to have a long memory, makes states “sticky” but prevents it from remaining permanently in any one state. Such a process also has a representation as a high order Markov chain, but one that is too complex to be represented easily as either a VLMC or an embedded first order model. It also allows for states far in the past to have a large influence over the transition to the next state, meaning that correctly modeling long range dependence is important here.

For each simulation, after the classes were generated the covariates were generated as bivariate Normals \(X_i \sim N(\mu_Y, \sigma^2)\) where the means of the three states were \(\mu_a = (2, 0)\), \(\mu_b = (0, 0)\), \(\mu_c = (0, 2)\) and \(\sigma\) varied from 0.1 to 6, depending on the simulation. This gives cases where the classes were almost completely separated and ones where they overlap greatly,
Figure 5: Structures of the two trees from the first group of simulations. At the leaves are
a next state probability distribution, where the ones for the top tree are given in figure
4 The bottom tree, “Tree 2” in our simulations, uses a similar set of functions to get its
transition probabilities.
which allows us to see the effect of noise on the performance of the sequential classifiers.

To evaluate the performance of different methods, on every member of the triple \((\tau, n, \sigma)\), where \(\tau\) is the process that generated the labels, we create 100 test sets and 100 training sets. A base classifier of either logistic regression or random forests, as well as a VLMC and 1st and 2nd order Markov Models were fit on each test set. On each test set we look at both the base classifiers as well as their output smoothed with each of the time series models and the true conditional class probabilities computed by using the true nonsequential class probabilities and the true time series model. Note, however, that for the third set of time series we did not compute true smoothed probabilities because the model does not admit a tractable representation as a first order Markov chain. The performance of each classifier + smoothing method is evaluated on misclassification as well as log loss and RMSE from the true model’s conditional class probabilities.

3.4.1. Simulation Results

We discuss the simulations in three qualitatively different groups based on the type of time series used to generate the labels. The first consists of the first order Markov Model and the two artificial trees, the second contains the simulations using large sparse trees based on fits of real data and the third group has the simulations generated from the “window” process discussed above.

**Group 1**

The first set of simulations is the least interesting, likely owing to the relatively simple structure of the trees used to generate the states. We show log loss and misclassification as seen in figures 6 and 7 respectively. For the first order model, we see as expected almost identical performance for smoothing with first or higher order chains in both log loss and misclassification loss. However, we do note that the performance depends highly on which classifier was used, with smoothed estimated based on logistic regression approaching the truth for all levels of noise and random forest based estimates, especially for high amounts
of noise, performing poorly even when smoothed. Smoothing of any sort does improve log loss more than misclassification loss, though this could be partially due to classifications that already had the correct modal probability being pushed toward 0 and 1.

For the other two chains, we observe a similar pattern in log and misclassification loss with the exception of a clear distinction between 1st order smoothing and either 2nd order or VLMC smoothing showing very similar performance, with VLMCs winning only by a small margin. Note that, again, all of the random forest-based classifiers underperform even the nonsequential logistic classifier as noise increases (though this is not true for the lowest nontrivial noise level). Further, for the logistic classifier both 2nd order and VLMC smoothing approach the truth as sample size increases, becoming especially close for high values of $\sigma$.

The RMSE plots for the first set of simulations seen in figure 8 show some similar patterns to the first two evaluations in that random forests consistently underperform. But, for the logistic base classifier the distance between the estimated and true probabilities increases as the noise becomes moderate and then decreases as it grows. This general pattern is likely due to the fact that our estimates are imperfect in the face of noise, but when the signal is comparatively weak, the true conditional class probabilities will all begin to converge to the marginal distribution, as will the estimated probabilities if our model is good.

Beyond this, we notice a few other patterns. First, the VLMC performs poorly for low $\sigma$, especially when it is close to 0. This appears in all of the simulations and sample sizes, though the effect is smaller when the tree is fit on more data. We suspect this is due to the more complex model having a bigger effect on probabilities that are already good and pushing them close to 0 and 1. Next, the second order model outperforms the VLMC on Tree 1, possibly because the VLMC is overfitting a relatively simple sequential structure. However, on Tree 2, where the structure is more complex, VLMC smoothing performs better.
Figure 6: Misclassification loss for the $1^{\text{st}}$ order Markov Chain and the two synthetic deep sparse trees. Notice that for the logistic classifier both the VLMC and second order model are close enough to the truth. Also note the large dependence on which initial classifier was used.
Figure 7: Log loss for the 1st order Markov Chain and the two synthetic deep sparse trees. Notice that for the logistic classifier both the VLMC and second order model are close enough to the truth. Also note the large dependence on which initial classifier was used.
Figure 8: RMSE of conditional class probabilities for the 1st order Markov Chain and the two synthetic deep sparse trees. Notice that for the logistic classifier the VLMC has difficulty estimating probabilities accurately unless the sample size is large. And even then, only for higher noise cases. Also note the poor performance when the initial probability estimates are bad with Random Forests.
Group 2

The next set of simulations is somewhat more interesting, and overall shows the effect that a more complicated underlying sequential model can have even when the emission distributions are the same. Note that sample sizes are restricted to only 2000 and 8000 due to the size of the trees used to generate the labels and the fact that one of the states was uncommon, leading to it not appearing in some length 500 simulations.

Several things jump out here compared to group 1. First, the difference between smoothing and not is noticeably larger. This applies to smoothing the true nonsequential conditional class probabilities with the true VLMC as well as to fit models. Second, random forests benefit more from smoothing for misclassification loss but still suffer in log loss for high amounts of noise. Third, when smoothing with the logistic classifier’s outputs all three models come close to approximating the truth again in both misclassification and log loss. However, there appears to be more of a difference between \( n = 2000 \) and 8000 than in the previous group of simulations. We also observe that both the 2\(^{nd}\) order and VLMC smoothing naturally produce estimates closer to the truth, but that the VLMC has a small but consistent advantage. This is especially apparent for the larger sample size and as \( \sigma \) increases.

The plots of RMSE (figure 11) also display a different pattern than those in the first group. We see three distinct groups — unsmoothed classifiers, smoothed random forest output, and smoothed logistic regression output. Next, as the noise increases and comparatively more information is being conveyed by the time series rather than covariates the VLMC shows gains over 1\(^{st}\) or 2\(^{nd}\) order smoothing. The difference is more noticeable for the logistic classifier, which again highlights that smoothing can only do so much with poor input probabilities. In addition to the difference, only the VLMC-smoothed logistic probabilities show the same decrease in RMSE for increasing noise that was evident in the first group of simulations. The gap for high values of \( \sigma \) is also noticeably larger, likely because more
sequential information is required to approximate the true model when the nonsequential probabilities become noisier.

**Group 3**

The final set of simulations is perhaps the most interesting because the $Y$-generating process only has a representation as a very high order Markov model. Even a VLMC should be at best able to capture only part of the structure since the entire past series of states up to the window length $w$ always matters even though the next state probabilities may be the same for many of them. Given this, we are not able to produce RMSE plots in this case since true model probabilities were not feasible to obtain. The misclassification and log loss plots for this set of simulations are shown in figures [12] and [13] respectively. As such, we only have the true nonsequential probabilities to compare against in these plots, though we will see that such comparisons are still interesting despite not having the theoretical lower bound for error.

As before, this group of simulations has three different sequential structures. What varies here is the length of the window that is looked back on to determine the next state probabilities. We consider windows of length 10, 20, and 40, finding qualitatively different behavior of our methods depending on the underlying model.

First, we address the misclassification loss shown in figure [12]. Notice that using VLMCs outperforms all other types of smoothing, with the gap increasing as the sample size and noise increase. It is apparent here the benefit of tree size being a function of the amount of data the VLMC was trained on; the bigger the tree the better a VLMC approximates the complex time series structure. However, it is interesting to note that for a sample size of 8000, while the gap between VLMC and second order smoothing increases as the window size goes from 10 to 20, it shrinks at 40. At this point, while all smoothing becomes increasingly problematic, the larger size of the VLMC is likely subject to fitting more erroneous patterns found in the data.
Figure 9: Misclassification loss for the large trees based on sleep data. Notice that for the logistic classifier the VLMC performs best when the sample size is larger, but other smoothed classifiers do well too. This is likely because one state is very rare. Note too the increased difference between classifier groups.
Figure 10: Log loss for the large trees based on sleep data. Notice that for the logistic classifier the VLMC performs best when the sample size is larger, but other smoothed classifiers do well too. This is likely because one state is very rare. Note too the increased difference between classifier groups.
Figure 11: RMSE of conditional class probabilities for the large trees based on sleep data. Here the Logistic classifier with VLMC smoothing clearly performs best except with low noise and improves with a larger sample size.
Another interesting observation is that, unlike in the previous groups, smoothing performs worse than the true nonsequential probabilities for $n = 500$, with the discrepancy increasing as the underlying sequential generating process becomes more complex. However, it still affords some benefits over whatever base classifier is being used.

As the sample size grows, we also see that smoothing starts to beat the unsmoothed truth and the difference between smoothed and unsmoothed classifiers increases. However, we cannot distinguish how much of the gain comes from a better base classifier and how much comes from a better estimate of the time series structure. There is reason to believe both play a role because the base classifiers clearly improve with sample size, but the difference between the VLMC and low order Markov models also increases since it should require a sample size to estimate the best first or second order approximation of the true sequential process.

For the log loss plots of the same group (figure 13) we see the same general pattern as with misclassification loss. The most immediately obvious difference, as expected, is that using random forests as a base classifier leads to notably worse relatively performance under log loss than under misclassification loss; although it does relatively poorly in both.

**Summary of Results**

The three groups of simulations highlight several facts about sequential learning. First, for the same emission probabilities we can get vastly different relative performance changes between smoothed and unsmoothed conditional class probability estimates depending on the complexity of the underlying structure on $Y_t$. When the time series structure was very complex, VLMCs showed a clear benefit under all measures of loss. But surprisingly, for simpler structures, even ones we thought would be complex enough that only a VLMC could perform well on, the low fixed order Markov models performed surprisingly well and were quite close to the VLMC. This was especially apparent when looking at misclassification loss, as the low order models may yield worse probability estimates, but all that matters in
Figure 12: Misclassification loss for the window process. Due to the complexity of this process, VLMC smoothing shows a clear benefit here, but only for larger sample sizes. The effect of sample size also increases as the process becomes more complex but that for the length 40 window some of the gains seem to diminish.
Figure 13: Log loss for the window process. Due to the complexity of this process, VLMC smoothing shows a clear benefit here, but only for larger sample sizes. The effect of sample size also increases as the process becomes more complex but that for the length 40 window some of the gains seem to diminish.
this case is being on the right side of the cutoff.

Second, the initial probability estimates matter for all measures of performance. Random forests, which were not as well suited to the data generating process as logistic regression, perform worse, especially on log loss and RMSE which directly penalize bad probability estimates even if the classes are correct. However, even with these poor initial estimates, smoothing does have a benefit relative to any base classifier under all measures of loss. On the other hand, even using the best sequential model with the poorer initial estimates frequently resulted in final estimates that were worse than unsmoothed ones using the better initial classifier. This was more apparent, as was the gap between all models using random forests and those using logistic regression as the simulation noise increased.
Chapter 4

Application: Mouse Sleep Data

In this chapter we revisit the mouse sleep stage dataset used in McShane et al. [39], applying sequential learning with VLMCs instead of a Generalized or Transition-Dependent Generalized Markov Model. We find that VLMCs can do as well as and in many cases better than these parametric models which were specifically selected for this data. This occurs in spite of the fact that no special tuning needed to be done for fitting VLMCs to the sequence of sleep states. Using sensible default values all yield similar results, demonstrating the model’s wide applicability. We also show how sequential learning with VLMCs can be improved by fitting a model on multiple time series. This is done using a modified context algorithm that recursively adds the trees grown on each series before pruning one large tree.

4.1. The Data

The dataset consists of the sleep stage as the class we are to predict along with covariates extracted from video of eight mice all of the strain C57BL/6J recorded for a period of 24 hours. The data is discretized into “epochs” where each epoch represents one 10-second period, yielding 8,640 observations for each mouse time series. Sleep stages consist of wake, non-REM, or REM sleep, the latter being of particular interest to sleep researchers because
of it is implicated as being important to functions such as memory consolidation [9, 26].

Detecting REM sleep also presents an additional challenge in our classification problem as REM is comparatively rare, occurring only about 5% of the time.

Sleep stages are determined by hand from observing electroencephalogram (EEG) and electromyogram (EMG) data taken from electrodes implanted in the mouse. Human scorers, however, do not always agree, with two scorers assigning different states to an epoch about 5% of the time [28]. This has prompted work on algorithms to automate the assignment of sleep states from EEG/EMG data such as that found in Sunagawa et al. [59], which uses spectral properties of the signal.

However, implanting the electrodes is still costly and time consuming, requiring mouse surgery and a recovery period of up to 14 days. Naturally this has led to interest in alternative methods of scoring sleep that are less expensive and invasive but can still come close to the accuracy of using EEG/EMG data. One popular method defined in Pack et al. [45] uses periods of 40 seconds or more of immobility distinguish sleep from wake and does well at this task but has no ability to distinguish REM from NREM sleep. As such we turn to the increasingly popular use of video tracking mice [22, 39] to distinguish between all three states. Our data in this chapter comes from McShane et al. [39].

The covariates for each epoch were computed from values obtained on the individual frames therein. Recall that each epoch is 10 seconds long and video was recorded at 10 frames per second. Tracking software was used to compute an ellipse approximating the shape and location of the mouse for each frame and then six values were computed for each 100 frame epoch. The values are: mean aspect ratio of the ellipse (ARM), intra-epoch standard deviation of the aspect ratio (ARsd), the mean and standard deviation of the size (area) of the ellipse, given in log units (LogSm and LogSsd), and the mean and standard deviation of the velocity, also in log units (LogVm and LogVsd). Also included is a binary variable indicating whether the lights were on (7AM-7PM) or off (7PM-7AM) in the mouse’s cage.

While we believe this and related studies would benefit from improved covariates, as we did
not have access to the original video, we only use the covariates from the original study. However, in our analysis we will use augmented covariates where we include in the analysis of one epoch the covariates or averages thereof from surrounding epochs. This includes an augmentation done in the original study and two new ones, one of which appears to yield a marginal improvement, strengthening the case for future studies with improved covariates.

Figure 14 shows estimates of the density of each of these covariates broken down by mouse and what stage of sleep the animal was in. We notice that wake can be fairly easily distinguished from sleep using these covariates, such as the obvious fact that awake mice move more. REM and NREM sleep on the other hand are harder to distinguish, with substantial overlap in their distributions. However some hope is apparent as was pointed out in McShane et al. [39] that the aspect ratio for mice in REM sleep tends to be lower than that of mice in NREM sleep. However, there is still significant overlap with both REM and wake. This provides some motivation for the use of sequential methods, as a lower aspect ratio may provide some evidence of REM, but if nearby states are very likely to be REM or even NREM we would want to increase this probability.

Figure 14 also highlights the presence of inter-mouse covariate variability. The distributions for a given covariate and sleep state are relatively consistent across mice but appear to vary enough that using a model fit on multiple mice would confer substantive benefit over one fit on only one mouse beyond just having access to more data for the fit.

In addition to differences between sleep states and mice, we see in figure 15 the difference in observed covariates depending on whether the light was on or off in the mouse cages. Note that this is only the difference in emission distributions and is not because the mice spend different amounts of time in each state, which we will discuss shortly. Also notice that the nature of the change is sometimes simply a shift in the distribution and other times a change in shape. However, regardless of sleep state, the changes in distribution appear to be relatively similar, so knowing about the light/dark difference may provide limited usefulness in practice.
Figure 14: Violin plot of the continuous covariates measured on the mice. Each panel is one variable, the X axis is for the individual mice and the colors represent which stage of sleep the mouse was in. The lines on the violins represent the 0.25 quantile, median and 0.75 quantile respectively and the black dot is the mean.
Figure 15: Violin plot of the continuous covariates measured on the mice. Each panel is one variable and one sleep state, the X axis is for the individual mice and the colors represent whether the lights were on or off. The lines on the violins represent the 0.25 quantile, median and 0.75 quantile respectively and the black dot is the mean.
In addition to inter-mouse variability in the observed covariates, we would also like to know if the mice differ in how long they spend in each sleep state both overall and by time of day. Table 1 gives the per-mouse and overall proportion of bouts spent in each state in total and broken down by lights on or off. First it shows that the amount of time spent in each state varies as a function of time of day, meaning that this predictor is likely useful, at least for distinguishing wake from sleep. But this is obvious since mice are nocturnal.

The tables also show mouse-to-mouse variability in the time spent in each state. One interesting part of this is that while the variation in wake vs. sleep is obvious we also see some variability in the ratio of REM to NREM sleep. Typically it is about a 1:10 ratio but we observe between 1:7 and 1:15 for some individual mice. Further, the magnitude of the differences can vary with the lights. All of this indicates we could benefit from pooling the mice to fit a model and it foreshadows a question that we will ask in the next chapter on how to handle situations where the marginal class distribution varies substantially from sample to sample.

Beyond the total amount of time that mice spend in each sleep state, researchers are also interested in the number of bouts of sleep or wakefulness a mouse has and how long each of these is. Figure 16 and table 2 provide plots and tables respectively for the distribution of bout lengths by mouse and sleep state. While we do not discuss in detail, previous work [38] has looked at tests of similarity between these distributions and found that some mouse pairs are significantly different though a large number are similar, which goes along with the intuition gleaned from visual inspection here.

Qualitatively, we see that bouts of REM are both shorter and less numerous than NREM as well as having smaller tails in the distributions. We also notice that much of the time a mouse spends awake is taken up by a few consolidated longer bouts. However, many of the wake bouts are extremely short, lasting only one or two epochs. These brief awakenings are of interest to sleep researchers [6] and may be worth trying to incorporate into a model in the future.
<table>
<thead>
<tr>
<th>State</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M8</th>
<th>M9</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>NREM</td>
<td>36.66</td>
<td>38.32</td>
<td>35.73</td>
<td>40.18</td>
<td>30.64</td>
<td>27.22</td>
<td>24.39</td>
<td>27.05</td>
<td>32.52</td>
</tr>
<tr>
<td>REM</td>
<td>4.40</td>
<td>3.61</td>
<td>2.50</td>
<td>3.22</td>
<td>3.36</td>
<td>3.22</td>
<td>3.15</td>
<td>3.70</td>
<td>3.39</td>
</tr>
<tr>
<td>wake</td>
<td>58.94</td>
<td>58.07</td>
<td>61.77</td>
<td>56.61</td>
<td>66.00</td>
<td>69.57</td>
<td>72.46</td>
<td>69.24</td>
<td>64.08</td>
</tr>
</tbody>
</table>

(a) Lights off

<table>
<thead>
<tr>
<th>State</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M8</th>
<th>M9</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>NREM</td>
<td>60.18</td>
<td>53.25</td>
<td>57.86</td>
<td>59.83</td>
<td>54.18</td>
<td>51.63</td>
<td>55.52</td>
<td>51.17</td>
<td>55.45</td>
</tr>
<tr>
<td>REM</td>
<td>5.58</td>
<td>5.90</td>
<td>3.64</td>
<td>6.81</td>
<td>8.10</td>
<td>5.81</td>
<td>6.27</td>
<td>7.94</td>
<td>6.26</td>
</tr>
<tr>
<td>wake</td>
<td>34.24</td>
<td>40.84</td>
<td>38.50</td>
<td>33.36</td>
<td>37.72</td>
<td>42.56</td>
<td>38.20</td>
<td>40.89</td>
<td>38.29</td>
</tr>
</tbody>
</table>

(b) Lights on

<table>
<thead>
<tr>
<th>State</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M8</th>
<th>M9</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>NREM</td>
<td>48.41</td>
<td>45.79</td>
<td>46.79</td>
<td>50.00</td>
<td>42.41</td>
<td>39.42</td>
<td>39.95</td>
<td>39.11</td>
<td>43.99</td>
</tr>
<tr>
<td>REM</td>
<td>4.99</td>
<td>4.76</td>
<td>3.07</td>
<td>5.01</td>
<td>5.73</td>
<td>4.51</td>
<td>4.71</td>
<td>5.82</td>
<td>4.82</td>
</tr>
<tr>
<td>wake</td>
<td>46.60</td>
<td>49.46</td>
<td>50.14</td>
<td>44.99</td>
<td>51.86</td>
<td>56.06</td>
<td>55.34</td>
<td>55.07</td>
<td>51.19</td>
</tr>
</tbody>
</table>

(c) Combined

Table 1: Tables on the percentage of time each mouse spends in each sleep state, broken down by time of day. The last column gives the average amount for all mice.

4.2. Analysis

We now turn to analysis of the data. We use sequential learning with several different sequential models as well as three possible base classifiers to get initial state probability estimates. In addition, we fit each base classifier using either only the video covariates provided or augmented covariates which include information from surrounding epochs.

The base classifiers we consider are Conditional Random Fields (CRF) as implemented by the Python implementation of CRFSuite [43], Random Forests, and multinomial Logistic regression. The covariate augmentations we consider are none (only using the video covariates provided), MA10(compute the moving averages of 10 lead and lag terms for each covariate), MA_5_10_15 (computing the moving averages for lead and lag 5, 10, and 15 for each covariate), and LeadLagAll10 (no moving averages, but use all 10 lead and lag terms
Figure 16: Bout durations by sleep state and mouse. Bouts of REM tend to be shorter and more evenly spread out than NREM and a few long bouts of wake account for most of it for each covariate). The purpose of the augmented covariates is both to see if using more long-range information benefits nonsequential methods, allowing them to perform similar to sequential ones, and to see if they do yield better initial estimates and this better starting point helps sequential methods as well. Note that we could not use LeadLagAll10 with CRFs due to the software running impossibly slowly and crashing with that many covariates.

The initial nonsequential probability estimates were smoothed with either nothing, a first order Markov model, a TDGMM, or a VLMC. For the VLMC-smoothed trials, we consider three cases where we vary the number of times a context must be observed to be included in the tree during fitting. We use the values 6, 12, and 18. But as expected, there is minimal variation in our results regardless of this value.

For each of the classifier, covariate augmentation, smoothing method cases, we consider training on one mouse and testing on every other and the “holdout” case where we train on seven of the mice and hold out one for testing. For smoothing with VLMCs in the holdout case, the base classifier was trained on the 7 training mice with no special preparation other than using augmented covariates when applicable and the VLMCs were trained by building a tree for each in-sample mouse and then adding them together prior to pruning using
Table 2: Bout statistics, including number and duration, broken down by sleep state and mouse as well as aggregated across mice.

algorithm This allows us to train a VLMC on many sequences of finite length easily. Otherwise we would have to paste the training sequences together, which could result in trees that are too large or poorer fits due to nontrivial influence of the points where the sequences are joined together if they are not particularly long relative to the size of the tree.

For the TDGMM, fitting on multiple sequences just requires converting all of the sequences to holding times individually and then pooling the holding times to estimate the duration distributions. However, we should note here that in the original work, the TDGMM was actually trained on all eight mice regardless of whether testing was being done with only one holdout mouse or on train-test pairs of mice. We retained this, though noting that it
may provide a slight advantage, especially when only one mouse should have been used to train the sequential part of the model.

In either case, both methods simply pool the data and do not model any kind of relationship between the mice explicitly. An example of this would be a hierarchical model, which could be especially useful if we were interested in inference on parameters for individual mice. However, doing so, especially for VLMCs where it is not clear what the parameters are is beyond the scope of this work, albeit a potentially interesting future direction.

**Algorithm 9 Add Context Trees**

**Require:** \( \tau_1 \) and \( \tau_2 \) share an alphabet \( A \)

**function** ADD\((\tau_1, \tau_2)\)

1. **if** Either \( \tau_1 \) or \( \tau_2 \) is empty **then**
2. **return** The nonempty tree
3. **end if**
4. **for** \( a \in A \) **do**
5. \( N_{\tau}(a) \leftarrow N_{\tau_1}(a) + N_{\tau_2}(a) \)
6. \( \tau\.children[a] \leftarrow \text{ADD}(\tau_1\.children[a], \tau_2\.children[a]) \)
7. **end for**
8. **return** \( \tau \)
9. **end function**

4.3. Results

Tables 3 and 4 give the overall misclassification rates for each combination of classifier, covariate augmentation and smoothing method for the holdout and mouse-by-mouse cases respectively. The first thing to notice is that with the exception of CRFs, training on multiple mice does better across by board by over 10% in most cases. This is not surprising as we know there is substantial inter-mouse variability both in the covariate distributions and the sequential properties of their sleep stages. As with the previous study, in both cases the nonsequential methods actually have lower misclassification rates than the sequential methods. This is due, as has been previously shown and we will discuss, to nonsequential models underpredicting REM sleep and sequential models overpredicting it.

However, notice that training on more mice decreases the gap between sequential and non-
sequential methods. This may be due to either better starting probability estimates or better models of the sequential part of the data. There is reason to believe the latter is at least part of the effect, though, as VLMCs go from underperforming to overperforming vs. the TDGMM. This could be from the fact that the TDGMM was always fit with more data or that a more complicated nonparametric model like the VLMC just requires more data to get a good fit. In either case, though, this speaks to the importance of having a good sequential model.

Since sequential models overpredict REM and overall perform worse because of it, we have to ask if this buys us anything. Tables 5 and 6 give the performance of each method, broken down by classifier, covariate augmentation and sequential smoothing when we train on all but one holdout mouse and when we train on one mouse and test on all others respectively. The columns of the tables provide, averaged over every out of sample mouse, the REM true positive, true negative, false positive, and false negative rates. Also provided are the positive and negative predictive values.

Generally, what we see is that training on multiple mice improves performance across the board and that all forms of smoothing overpredict REM. The positive predictive value,
Table 4: Comparison of misclassification rates training on one mouse and testing on every other.

which is the probability that a state we called REM actually was REM caps out at around 31%. While this number can get into the 60’s for some nonsequential methods with augmented covariates, the true positive rate is far lower. Since REM is particularly important to sleep researchers, we would prefer overpredicting to underpredicting. Also, the fact that sequential methods are able to find REM is very promising because we believe that with cleaner initial data we could retain the detection of REM while reducing the false positive rate. Nonsequential methods show less hope in this regard. Finally, depending on which base classifier and covariate augmentation was used, either VLMCs and TDGMMs perform better, but generally have similar performance.

Next we examine ROC curves for the holdout estimates (per-mouse are left out for economy of space, but have similar qualitative properties and worse overall performance). Figures 17, 18, and 19 give the ROC curves and AUCs broken down by covariate augmentation, base classifier and type of smoothing used.

We see that for wake, performance is more or less similar regardless of the method used. The only place unsmoothed methods suffer is if there are no augmented covariates, where

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Augmentation</th>
<th>Smoothing method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>none</td>
<td>IMM</td>
</tr>
<tr>
<td>CRF</td>
<td>MA_5,10,15</td>
<td>0.329</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>0.134</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>0.148</td>
</tr>
<tr>
<td>Logistic</td>
<td>LeadLagAll10</td>
<td>0.138</td>
</tr>
<tr>
<td></td>
<td>MA_5,10,15</td>
<td>0.126</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>0.126</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>0.149</td>
</tr>
<tr>
<td>RandomForest</td>
<td>LeadLagAll10</td>
<td>0.134</td>
</tr>
<tr>
<td></td>
<td>MA_5,10,15</td>
<td>0.126</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>0.128</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>0.161</td>
</tr>
</tbody>
</table>
all forms of smoothing provide a similar boost. For NREM, we actually see unsmoothed classifiers outperforming the smoothed ones in most cases, with first order Markov models doing the worst and the more complex models tending to clump together. Also of note here is that the performance for CRFs, while usually bad is unusually awful when using three moving averages for covariate augmentation. This speaks to the poor quality and instability of results we can get from this model.

Finally, for REM, we have expectedly worse performance than for the other two states overall. Smoothing also tends to outperform not smoothing as well as smoothing with a first order model, and VLMCs usually perform the best, especially when no augmented covariates are used. This again points to the value of sequential methods, as being correct on REM is very often more important than other states to sleep researchers.

4.3.1. Two Stage Classification

In addition to the stated problem of distinguishing the wake state from both REM and NREM sleep, sleep researchers are often interested in just distinguishing sleep from wake. The Tables 8 and 7 give the misclassification rates for each of the method, variable augmentation, smoothing triples previously discussed for holdout and mouse pair evaluations respectively. Since we are only trying to tell sleep from wake, other than EEG, the method to beat here is the 40-second rule defined in Pack et al. [45] which declares a mouse to be asleep using video data if it spends more than 40 seconds moving at a speed of less than 3 pixels/second.

On this dataset, the 40-second rule achieves an overall error rate of 7.77%. So, the first thing to notice from our tables is that if we train on only one mouse, using lagged covariates hovers around this value if we use logistic regression or random forests as our base classifier. This is true regardless of whether or not we smooth, with all smoothing helping anywhere from not at all to an improvement of about half a percentage point. On the other hand, if we use no lagged covariates, the different smoothing methods offer a percentage point or
Figure 17: ROC curves with AUC for REM broken down by nonsequential model and covariate augmentation. Both sequential and nonsequential methods do a reasonable job when augmented covariates are provided. What is most interesting is that the only place VLMCs have a clear advantage over everything else is when no augmentation is used.
Figure 18: ROC curves with AUC for non-REM broken down by nonsequential model and covariate augmentation. Both sequential and nonsequential methods do well regardless of the covariates provided and the nonsequential methods tend to be best. The CRF+MA_5_10_15 panel highlights how wild “smoothing” can make performance when the initial estimates are very bad.
Figure 19: ROC curves with AUC for wake broken down by nonsequential model and covariate augmentation. Both sequential and nonsequential methods do very well regardless of the covariates provided with everything being almost equal when they are augmented. Nothing particularly stands out here as awake versus not awake is a much easier problem.
two of improvement, but still underperform this simple rule.

However, table 8 which shows the performance training on multiple mice and testing on one holdout tells a different story. While smoothing of any kind still lowers the misclassification rate by between one and two percentage points when no augmented covariates are used, providing augmented covariates in the form of preceding and following epochs’ values or moving averages thereof changes the situation dramatically.

Using Random Forest with length 5, 10, and 15 moving averages of the video covariates, we achieve a 6% misclassification rate. This is approaching the level of inter-annotator disagreement. Also, smoothing when performance is at or near this level, as the case may be with other methods used, yields worse overall performance. For the task of distinguishing sleep from wakefulness in mice using video, it does not appear that modeling the sequential nature of sleep is helpful, at least starting from the covariates we have. Of course, the video itself is low resolution and the features are simple, so there is reason to believe that we can improve further on both the nonsequential and sequential front by using better data and similar models to create better initial estimates that have lower initial error but also benefit from accurate sequential models.

4.3.2. Probability Estimation

Next we consider the quality of the probability estimates for each state produced by the classifiers. Figures 20 and 21 give probability calibration plots for the case where we train on one mouse and train on seven mice respectively. The most obvious standout is that smoothing of any sort does not produce improved calibration. In fact, it appears to render any probability not close to 0 or 1 near meaningless as the middle of the range is relatively flat regardless of the sleep stage being looked at. Qualitatively, we see little difference regardless of what sequential model is being used.

In addition to this “un-calibration” effect, we also notice that using multiple mice to fit produces more calibrated initial probability estimates for both classifiers considered here.
However, they still perform poorly on REM unless we augment the covariates with those from the surrounding epochs. While there is improvement in both cases, when training on multiple mice, the REM curves come very close to those for NREM and wake.

Figure 20: Calibration plot for training on one mouse and testing on every other with two covariate augmentations, two classifiers and four types of smoothing. Colors represent sleep stage. The x axis is the predicted probability and the y axis is how often estimates in that bin for a given label turned out to have that label. Initial, unsmoothed estimates are moderately well calibrated, with NREM being underpredicted and REM and wake overpredicted. REM is particularly bad. Smoothing has a flattening effect on the calibration curves.

Does Lack of Calibration Matter?

Given what we saw with sequential methods yielding conditional class probabilities that are calibrated worse than the initial nonsequential estimates, we would like to ask why their performance using standard assessments was not worse. However, while this bodes poorly for our ability to generate accurate probability estimates, it matters less for misclassifying states, as all of the probability estimates are pushed toward 0 or 1. Figure 22 shows the
Figure 21: Calibration plot for training on seven mice and testing on one holdout with two covariate augmentations, two classifiers and four types of smoothing. Colors represent sleep stage. The x axis is the predicted probability and the y axis is how often estimates in that bin for a given label turned out to have that label. Unsmoothed estimates are much better calibrated, especially for REM and especially using covariate augmentation. However, applying sequential methods still flattens out the curves.
distribution of probability estimates for each state by method used. In every case we see the same general phenomenon that regardless of what the distribution looks like initially, all of the mass goes toward the edges. While this certainly needs assessment in the future, it means that the lack of calibration won’t necessarily hurt us when using sequential methods for just trying to pick a modal class.

4.3.3. Bout Length Considerations

In addition to just assessing probability estimates and misclassification rates, we would like some measure of performance on the sequential nature of this data. As discussed, sleep researchers are often interested in the amount of time spent in a given state as well as the number of bouts of sleep or wakefulness and how long each lasts. Table 9 shows bout length statistics for each of our methods considered on the holdout fits. The general pattern is that sequential methods tend to be too “sticky” meaning that they estimate fewer, longer bouts for each state. The only place we do not see this is with augmented covariates and REM sleep, where both the number and duration of the bouts are too large.

A potential explanation for this “over-smoothing” is that sequential methods are doing unusually poorly on the short duration bouts. To investigate this we return to the brief awakenings we previously discussed where a mouse wakes up for a short period of time in the middle of a bout of sleep. While only 2.4% of the time spent awake is in bouts of length 1 and 2.3% is in bouts of length 2, around 45% of all waking bouts are of length 1 and 21.5% are of length 2. This means there are many opportunities for a small number of mistakes to negatively impact our bout statistics.

Table[10] shows that both covariate augmentation and smoothing hurt our performance on length 1 bouts of wake. In fact, any type of smoothing generally results in an error rate of 80-90%. While not shown, we see similar a similar effect, albeit with slightly lower error rates on wake bouts of length 2. The implication here is that better detection of short bouts of wake and sleep may provide a large benefit when the quantity of interest is based on the
Figure 22: Histograms of estimated class probabilities. Sequential methods move all probability estimates toward the boundary about equally regardless of the classifier or covariate augmentation used.
lengths of sleep/wake bouts.

These types of errors point in an important next direction. It is known that genetics can influence sleep consolidation but not the total amount of sleep in mice [24], so modeling the number and duration of bouts is important to the scientific enterprise. However, it may not be necessary to use a method that accurately and precisely measures these quantities, as the question of interest may be if certain strains of mice or mice treated with a specific drug exhibit increases or decreases in the quantity and consolidation of sleep and wakefulness. What would matter in this case is whether changes in these quantities are accurately reflected in a model’s output. That is, would it tell us that strains that consolidate sleep more have fewer, longer bouts of sleep than those that do not. This is a question for future work and potentially useful if the answer is yes.
<table>
<thead>
<tr>
<th>Classifier</th>
<th>Augmentation</th>
<th>Smoothing</th>
<th>R tp</th>
<th>R tn</th>
<th>R fp</th>
<th>R fn</th>
<th>R ppp</th>
<th>R npv</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF</td>
<td>MA10</td>
<td>1MM</td>
<td>0.536</td>
<td>0.908</td>
<td>0.092</td>
<td>0.464</td>
<td>0.228</td>
<td>0.975</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>TDGMM</td>
<td>0.539</td>
<td>0.912</td>
<td>0.088</td>
<td>0.461</td>
<td>0.238</td>
<td>0.975</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{12}</td>
<td>0.543</td>
<td>0.907</td>
<td>0.093</td>
<td>0.457</td>
<td>0.228</td>
<td>0.975</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{6}</td>
<td>0.537</td>
<td>0.913</td>
<td>0.087</td>
<td>0.463</td>
<td>0.238</td>
<td>0.975</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>1MM</td>
<td>0.110</td>
<td>0.992</td>
<td>0.008</td>
<td>0.890</td>
<td>0.398</td>
<td>0.956</td>
</tr>
<tr>
<td>LeadLagAll10</td>
<td>MA10</td>
<td>1MM</td>
<td>0.846</td>
<td>0.638</td>
<td>0.362</td>
<td>0.154</td>
<td>0.106</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>TDGMM</td>
<td>0.842</td>
<td>0.644</td>
<td>0.356</td>
<td>0.158</td>
<td>0.107</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{12}</td>
<td>0.840</td>
<td>0.646</td>
<td>0.354</td>
<td>0.160</td>
<td>0.108</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{6}</td>
<td>0.841</td>
<td>0.644</td>
<td>0.356</td>
<td>0.159</td>
<td>0.107</td>
<td>0.988</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>1MM</td>
<td>0.764</td>
<td>0.689</td>
<td>0.311</td>
<td>0.236</td>
<td>0.111</td>
<td>0.983</td>
</tr>
<tr>
<td>Logistic</td>
<td>MA10</td>
<td>1MM</td>
<td>0.638</td>
<td>0.909</td>
<td>0.091</td>
<td>0.362</td>
<td>0.262</td>
<td>0.980</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>TDGMM</td>
<td>0.638</td>
<td>0.918</td>
<td>0.082</td>
<td>0.362</td>
<td>0.283</td>
<td>0.980</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{12}</td>
<td>0.660</td>
<td>0.920</td>
<td>0.080</td>
<td>0.340</td>
<td>0.295</td>
<td>0.982</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{6}</td>
<td>0.655</td>
<td>0.919</td>
<td>0.081</td>
<td>0.345</td>
<td>0.291</td>
<td>0.981</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>1MM</td>
<td>0.701</td>
<td>0.909</td>
<td>0.091</td>
<td>0.299</td>
<td>0.281</td>
<td>0.984</td>
</tr>
<tr>
<td>RandomForest</td>
<td>LeadLagAll10</td>
<td>1MM</td>
<td>0.644</td>
<td>0.905</td>
<td>0.095</td>
<td>0.356</td>
<td>0.256</td>
<td>0.980</td>
</tr>
<tr>
<td></td>
<td>LeadLagAll10</td>
<td>TDGMM</td>
<td>0.652</td>
<td>0.918</td>
<td>0.082</td>
<td>0.348</td>
<td>0.289</td>
<td>0.981</td>
</tr>
<tr>
<td></td>
<td>LeadLagAll10</td>
<td>VLMC_{12}</td>
<td>0.663</td>
<td>0.923</td>
<td>0.077</td>
<td>0.337</td>
<td>0.301</td>
<td>0.982</td>
</tr>
<tr>
<td></td>
<td>LeadLagAll10</td>
<td>VLMC_{6}</td>
<td>0.662</td>
<td>0.918</td>
<td>0.082</td>
<td>0.338</td>
<td>0.290</td>
<td>0.982</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>1MM</td>
<td>0.664</td>
<td>0.921</td>
<td>0.079</td>
<td>0.336</td>
<td>0.299</td>
<td>0.982</td>
</tr>
<tr>
<td>RandomForest</td>
<td>MA10</td>
<td>1MM</td>
<td>0.701</td>
<td>0.909</td>
<td>0.091</td>
<td>0.299</td>
<td>0.281</td>
<td>0.984</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>TDGMM</td>
<td>0.693</td>
<td>0.921</td>
<td>0.079</td>
<td>0.307</td>
<td>0.309</td>
<td>0.983</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{12}</td>
<td>0.699</td>
<td>0.920</td>
<td>0.080</td>
<td>0.307</td>
<td>0.307</td>
<td>0.984</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{6}</td>
<td>0.694</td>
<td>0.921</td>
<td>0.079</td>
<td>0.306</td>
<td>0.310</td>
<td>0.983</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>1MM</td>
<td>0.701</td>
<td>0.909</td>
<td>0.091</td>
<td>0.299</td>
<td>0.281</td>
<td>0.984</td>
</tr>
<tr>
<td>RandomForest</td>
<td>LeadLagAll10</td>
<td>1MM</td>
<td>0.430</td>
<td>0.851</td>
<td>0.149</td>
<td>0.570</td>
<td>0.128</td>
<td>0.967</td>
</tr>
<tr>
<td></td>
<td>LeadLagAll10</td>
<td>TDGMM</td>
<td>0.413</td>
<td>0.895</td>
<td>0.105</td>
<td>0.587</td>
<td>0.167</td>
<td>0.968</td>
</tr>
<tr>
<td></td>
<td>LeadLagAll10</td>
<td>VLMC_{12}</td>
<td>0.417</td>
<td>0.912</td>
<td>0.088</td>
<td>0.583</td>
<td>0.194</td>
<td>0.969</td>
</tr>
<tr>
<td></td>
<td>LeadLagAll10</td>
<td>VLMC_{6}</td>
<td>0.417</td>
<td>0.911</td>
<td>0.089</td>
<td>0.583</td>
<td>0.192</td>
<td>0.968</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>1MM</td>
<td>0.003</td>
<td>0.999</td>
<td>0.001</td>
<td>0.997</td>
<td>0.229</td>
<td>0.952</td>
</tr>
<tr>
<td>RandomForest</td>
<td>MA10</td>
<td>1MM</td>
<td>0.754</td>
<td>0.841</td>
<td>0.159</td>
<td>0.246</td>
<td>0.194</td>
<td>0.985</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>TDGMM</td>
<td>0.742</td>
<td>0.867</td>
<td>0.133</td>
<td>0.258</td>
<td>0.220</td>
<td>0.985</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{12}</td>
<td>0.737</td>
<td>0.874</td>
<td>0.126</td>
<td>0.263</td>
<td>0.229</td>
<td>0.985</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{6}</td>
<td>0.740</td>
<td>0.867</td>
<td>0.133</td>
<td>0.260</td>
<td>0.221</td>
<td>0.985</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>1MM</td>
<td>0.735</td>
<td>0.875</td>
<td>0.125</td>
<td>0.265</td>
<td>0.229</td>
<td>0.985</td>
</tr>
<tr>
<td>RandomForest</td>
<td>MA10</td>
<td>1MM</td>
<td>0.710</td>
<td>0.856</td>
<td>0.144</td>
<td>0.290</td>
<td>0.200</td>
<td>0.983</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>TDGMM</td>
<td>0.731</td>
<td>0.867</td>
<td>0.133</td>
<td>0.269</td>
<td>0.219</td>
<td>0.984</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{12}</td>
<td>0.704</td>
<td>0.881</td>
<td>0.119</td>
<td>0.296</td>
<td>0.231</td>
<td>0.983</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>VLMC_{6}</td>
<td>0.712</td>
<td>0.874</td>
<td>0.126</td>
<td>0.288</td>
<td>0.233</td>
<td>0.984</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>1MM</td>
<td>0.708</td>
<td>0.880</td>
<td>0.120</td>
<td>0.292</td>
<td>0.231</td>
<td>0.983</td>
</tr>
<tr>
<td>none</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Performance of the various methods for the rare, hard to detect, and important REM state of sleep training on seven mice and holding out one for testing.
<table>
<thead>
<tr>
<th>Classifier</th>
<th>Augmentation</th>
<th>Smoothing</th>
<th>$R_{tp}$</th>
<th>$R_{tn}$</th>
<th>$R_{fp}$</th>
<th>$R_{fn}$</th>
<th>$R_{ppv}$</th>
<th>$R_{npv}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF</td>
<td>MA10</td>
<td>1MM</td>
<td>0.540</td>
<td>0.878</td>
<td>0.122</td>
<td>0.460</td>
<td>0.184</td>
<td>0.974</td>
</tr>
<tr>
<td></td>
<td>TDGMM</td>
<td>0.557</td>
<td>0.888</td>
<td>0.112</td>
<td>0.443</td>
<td>0.202</td>
<td>0.973</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC$_{12}$</td>
<td>0.561</td>
<td>0.884</td>
<td>0.116</td>
<td>0.439</td>
<td>0.198</td>
<td>0.975</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC$_{6}$</td>
<td>0.555</td>
<td>0.888</td>
<td>0.112</td>
<td>0.445</td>
<td>0.201</td>
<td>0.975</td>
<td></td>
</tr>
<tr>
<td>none</td>
<td>none</td>
<td>0.210</td>
<td>0.980</td>
<td>0.020</td>
<td>0.790</td>
<td>0.345</td>
<td>0.961</td>
<td></td>
</tr>
<tr>
<td>TDGMM</td>
<td>MA5$_{10}$</td>
<td>1MM</td>
<td>0.741</td>
<td>0.693</td>
<td>0.307</td>
<td>0.259</td>
<td>0.109</td>
<td>0.971</td>
</tr>
<tr>
<td></td>
<td>TDGMM</td>
<td>0.740</td>
<td>0.701</td>
<td>0.299</td>
<td>0.260</td>
<td>0.112</td>
<td>0.982</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC$_{12}$</td>
<td>0.742</td>
<td>0.701</td>
<td>0.299</td>
<td>0.258</td>
<td>0.112</td>
<td>0.982</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC$_{6}$</td>
<td>0.742</td>
<td>0.699</td>
<td>0.301</td>
<td>0.258</td>
<td>0.112</td>
<td>0.982</td>
<td></td>
</tr>
<tr>
<td>none</td>
<td>none</td>
<td>0.653</td>
<td>0.747</td>
<td>0.253</td>
<td>0.347</td>
<td>0.116</td>
<td>0.977</td>
<td></td>
</tr>
<tr>
<td>Logistic</td>
<td>LeadLagAll10</td>
<td>1MM</td>
<td>0.594</td>
<td>0.900</td>
<td>0.100</td>
<td>0.406</td>
<td>0.232</td>
<td>0.978</td>
</tr>
<tr>
<td></td>
<td>TDGMM</td>
<td>0.597</td>
<td>0.907</td>
<td>0.093</td>
<td>0.403</td>
<td>0.246</td>
<td>0.976</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC$_{12}$</td>
<td>0.608</td>
<td>0.905</td>
<td>0.095</td>
<td>0.392</td>
<td>0.245</td>
<td>0.978</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC$_{6}$</td>
<td>0.608</td>
<td>0.904</td>
<td>0.096</td>
<td>0.394</td>
<td>0.244</td>
<td>0.978</td>
<td></td>
</tr>
<tr>
<td>none</td>
<td>none</td>
<td>0.607</td>
<td>0.905</td>
<td>0.095</td>
<td>0.393</td>
<td>0.245</td>
<td>0.978</td>
<td></td>
</tr>
<tr>
<td>RandomForest</td>
<td>LeadLagAll10</td>
<td>1MM</td>
<td>0.648</td>
<td>0.902</td>
<td>0.098</td>
<td>0.413</td>
<td>0.232</td>
<td>0.977</td>
</tr>
<tr>
<td></td>
<td>TDGMM</td>
<td>0.597</td>
<td>0.907</td>
<td>0.093</td>
<td>0.403</td>
<td>0.246</td>
<td>0.976</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC$_{12}$</td>
<td>0.608</td>
<td>0.905</td>
<td>0.095</td>
<td>0.392</td>
<td>0.245</td>
<td>0.978</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC$_{6}$</td>
<td>0.608</td>
<td>0.904</td>
<td>0.096</td>
<td>0.394</td>
<td>0.244</td>
<td>0.978</td>
<td></td>
</tr>
<tr>
<td>none</td>
<td>none</td>
<td>0.607</td>
<td>0.905</td>
<td>0.095</td>
<td>0.393</td>
<td>0.245</td>
<td>0.978</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Performance of the various methods for the rare, hard to detect, and important REM state of sleep training on one mouse and testing on each other one.
Table 7: Comparison of two state misclassification rates holding one mouse out for testing. The 40-second rule calibrated on this data set achieves a 7.77% error rate.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Augmentation</th>
<th>Smoothing method</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF</td>
<td>MA_5_10_15</td>
<td>none 0.098</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>1MM 0.105</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>TDGMM 0.102</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_6 0.101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_12 0.101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_18 0.101</td>
</tr>
<tr>
<td>Logistic</td>
<td>LeadLagAll10</td>
<td>none 0.090</td>
</tr>
<tr>
<td></td>
<td>MA_5_10_15</td>
<td>1MM 0.084</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>TDGMM 0.081</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>VLMC_6 0.079</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_12 0.079</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_18 0.079</td>
</tr>
<tr>
<td>RandomForest</td>
<td>LeadLagAll10</td>
<td>none 0.082</td>
</tr>
<tr>
<td></td>
<td>MA_5_10_15</td>
<td>1MM 0.075</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>TDGMM 0.076</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>VLMC_6 0.076</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_12 0.076</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_18 0.076</td>
</tr>
</tbody>
</table>

Table 8: Comparison of two state misclassification rates holding one mouse out for testing. The 40-second rule calibrated on this data set achieves a 7.77% error rate.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Augmentation</th>
<th>Smoothing method</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF</td>
<td>MA_5_10_15</td>
<td>none 0.080</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>1MM 0.079</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>TDGMM 0.078</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_6 0.078</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_12 0.078</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_18 0.078</td>
</tr>
<tr>
<td>Logistic</td>
<td>LeadLagAll10</td>
<td>none 0.075</td>
</tr>
<tr>
<td></td>
<td>MA_5_10_15</td>
<td>1MM 0.076</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>TDGMM 0.070</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>VLMC_6 0.070</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_12 0.070</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_18 0.070</td>
</tr>
<tr>
<td>RandomForest</td>
<td>LeadLagAll10</td>
<td>none 0.070</td>
</tr>
<tr>
<td></td>
<td>MA_5_10_15</td>
<td>1MM 0.069</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>TDGMM 0.067</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>VLMC_6 0.067</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_12 0.067</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VLMC_18 0.067</td>
</tr>
<tr>
<td>state</td>
<td>covariates</td>
<td>smoothing</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>N</td>
<td>MA_5,10,15</td>
<td>DATA</td>
</tr>
<tr>
<td></td>
<td>DATA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1MM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TDGMM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1MM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TDGMM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1MM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TDGMM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1MM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TDGMM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VLMC,6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

Table 9: Bout statistics for fits using logistic base estimates and two types of covariate augmentation for fits using seven training and one holdout mouse. Qualitatively, other base classifiers and covariate augmentations are similar, as are the mouse pair fits. The general pattern we see is that smoothing yields too few bouts that are too long for all of the states. It may be losing short bouts of a different state interspersed in a longer period where the mouse is mostly in another. Statistics from the actual data are given on the first row for each state.
<table>
<thead>
<tr>
<th>Classifier</th>
<th>Augmentation</th>
<th></th>
<th>1MM</th>
<th>TDGMM</th>
<th>VLMC.6</th>
<th>VLMC.12</th>
<th>VLMC.18</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRF</td>
<td>MA5_10_15</td>
<td>0.87</td>
<td>0.89</td>
<td>0.89</td>
<td>0.89</td>
<td>0.89</td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>0.85</td>
<td>0.88</td>
<td>0.86</td>
<td>0.86</td>
<td>0.86</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>0.86</td>
<td>0.89</td>
<td>0.87</td>
<td>0.88</td>
<td>0.88</td>
<td>0.88</td>
</tr>
<tr>
<td>Logistic</td>
<td>LeadLagAll10</td>
<td>0.88</td>
<td>0.91</td>
<td>0.89</td>
<td>0.90</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td>MA5_10_15</td>
<td>0.67</td>
<td>0.89</td>
<td>0.85</td>
<td>0.85</td>
<td>0.86</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>0.67</td>
<td>0.87</td>
<td>0.81</td>
<td>0.83</td>
<td>0.84</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>0.47</td>
<td>0.88</td>
<td>0.81</td>
<td>0.82</td>
<td>0.82</td>
<td>0.83</td>
</tr>
<tr>
<td>RandomForest</td>
<td>LeadLagAll10</td>
<td>0.88</td>
<td>0.90</td>
<td>0.89</td>
<td>0.90</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td></td>
<td>MA5_10_15</td>
<td>0.53</td>
<td>0.90</td>
<td>0.86</td>
<td>0.86</td>
<td>0.87</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>MA10</td>
<td>0.52</td>
<td>0.89</td>
<td>0.85</td>
<td>0.86</td>
<td>0.86</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td>0.43</td>
<td>0.86</td>
<td>0.80</td>
<td>0.82</td>
<td>0.82</td>
<td>0.82</td>
</tr>
</tbody>
</table>

Table 10: Error rates given that the mouse was in a length 1 bout of wake.
In this chapter, we begin by revisiting a simulation study from McShane [38], first replicating the original results and then following on with an extension that explains and resolves the discrepancy in the original. The simulations studied sequential classification assuming either the state transitions $P(Y_{t+1} | Y_{0:t})$ or the conditional class probabilities $P(Y_t | X_t)$ were known and the other estimated. They were initially surprising because it was found that knowing one of these led to worse performance than estimating both. However, we find that at least some of this discrepancy can be resolved by understanding the role knowledge of the marginal distribution $P(Y_t)$ has in sequential learning. We then discuss more generally how knowledge of the marginal distribution of class labels can be used in classification algorithms.

5.1. Original Simulations

Two time series structures on $(Y_t)$ were considered. Both followed Generalized Markov Models; one with a finite holding time distribution for states and one with an infinite distribution. We have a multiclass problem with $\mathcal{A} = \{a, b, c\}$. For the GMM with finite
holding times, they followed a discrete beta distribution whose PMF is defined by

\[ P(H = i) = f_{\alpha, \beta} \left( \frac{1}{2M} + \frac{i - 1}{M} \right) \]

where \( f_{\alpha, \beta} \) is a Beta PMF with parameters \( \alpha \) and \( \beta \) and \( M \) is a positive integer. For the GMM with unbounded holding times, they were modeled by a Beta-Negative Binomial random variable with a Geometric tail which has the PMF

\[ P(H = i) = \frac{q}{c_M} I_{H \leq M} f_{\alpha, \beta, r}(i - 1) + (1 - q) + I_{H > M} (g_p(i - M)) \]

where \( f_{\alpha, \beta, r}(k) = \frac{\Gamma(r+k)}{k! \Gamma(r)} \frac{B(\alpha+r, \beta+k)}{B(\alpha, \beta)} \) is a Beta negative binomial PMF \(^{[31]}\) and \( g_p(k) = (1 - p)^{k-1}p \) is a geometric PMF. The parameter \( M \) determines how much of the holding time distribution comes from the Beta negative binomial. Recall that the geometric tail is necessary to represent the GMM as a Markov transition matrix when the holding times are unbounded.

The emission distributions were given by \( X_t | Y_t = i \sim \mathcal{N}(\mu_i, \sigma^2) \). For the simulations we fix \( \mu_a = 0, \mu_b = 1, \mu_c = 2 \) and let \( \sigma \) vary from 0.01 to 10 to assess the effect of noise on classifier performance.

For the time series parameters we let \( M \) vary from 3 to 10. The transition and holding time parameters are given in \(^{[23]}\) for the sake of completeness.

(a) Transition matrix

\[
\begin{pmatrix}
0 & 0.5 & 0.5 \\
0.75 & 0 & 0.25 \\
1/3 & 2/3 & 0
\end{pmatrix}
\]

(b) Holding time parameters (Discrete Beta)

\[
\begin{align*}
\alpha & \quad \beta \\
a & \quad 1 & \quad 1 \\
b & \quad e^{-15} & \quad 1 \\
c & \quad e^{0.75} & \quad e^{1.5}
\end{align*}
\]

(c) Holding time parameters (BNBD w/ tail)

\[
\begin{align*}
\alpha & \quad \beta & \quad r & \quad q & \quad p \\
a & \quad 1 & \quad e^5 & \quad 1 & \quad \frac{1}{1+e^{-1}} \\
b & \quad e^{-40} & \quad 1 & \quad 1 & \quad \frac{1}{1+e^{-1}} \\
c & \quad e^1 & \quad e^2 & \quad 1 & \quad \frac{1}{1+e^{-1}} \quad \frac{1}{1+e^{-1}}
\end{align*}
\]

Figure 23: Parameters for GMM simulations.

For each value of \( M \) and \( \sigma \) considered, 100 simulations were run for each of two sequence

79
lengths $T$; $T = 1000$ and $T = 10000$. The test data for each simulation was 200 points continuing on from the original $T$ points.

5.1.1. Performance Assessment

On each training set, a GMM was fit according to the parametric family from which it was generated and a nonsequential model for $\hat{P}(Y|X)$ was fit with simple logistic regression. Then, conditional class probabilities, $\hat{P}(Y_i|X_{(T+1):(T+200)})$ for $i \in T+1, \ldots, T+200$, for each state in the test set were computed using the discriminative Forward-Backward algorithm in each of four cases:

1. Using the true parameters of the GMM and the true emission distribution, which when combined with the marginal distribution gives the true conditional class probabilities. All of the others will be compared against this.

2. Using the estimated GMM and the estimated nonsequential class probabilities.

3. Using the estimated GMM and the true nonsequential conditional class probabilities.

4. Using the true GMM and the estimated nonsequential class probabilities.

We refer to cases 3 and 4 above as semi-oracles because they each have access to part of the true model: either the nonsequential emission distribution $f(X_t|Y_t = i)$ and conditional class probabilities $P(Y_t | X_t)$ or the Markov process that generated the underlying states.

The performance of cases 2 through 4 was assessed by computing the root mean square difference of the probability estimates between it and the true probabilities from case 1 and averaging over all 100 runs. So, we would have

$$\sqrt{\frac{1}{100} \sum_{n=1}^{100} \frac{1}{200} \sum_{t=T+1}^{T+200} \sum_{i \in \mathcal{A}} (P(Y_t^{(n)} = i \mid X_t^{n}_{(T+1):(T+200)}) - \hat{P}(Y_t^{(n)} = i \mid X_t^{n}_{(T+1):(T+200)})^2}$$
where \( Y_t^{(n)} \) is the \( t \)th observation in simulation number \( n \) and \( \hat{P}^{(n)} \) is the estimated conditional class probability function on simulation \( n \).

Now comes the surprising part. It was found that in most cases, estimating everything outperformed both semi-oracles as can be seen in figures 24 and 25.

While the overall error rates are lower for \( T = 10000 \) training points than for \( T = 1000 \) training points we tend to see the same general patterns. When the holding times are distributed as discrete Betas, and the time series structure is simpler (smaller values of \( M \)) estimating the Markov model and the nonsequential class probabilities performs the worst or similarly to only knowing the time series regardless of how noisy the data is. As \( M \) increases, however, it starts to outperform both semi-oracles, especially at high noise levels. Knowing the true nonsequential conditional class probabilities, however, continues to do well when \( \sigma \) is small, which is likely due to the fact that the conditional class probabilities start off closer to 0 and 1 and are moved less by the time series. Also, note that knowledge of the CCPs almost uniformly beats knowledge of the GMM, except when \( M \) is small and the noise is very low.

When the holding times come from a Beta Negative-Binomial with a Geometric tail, we see the completely estimated model performing best at all values of \( M \). We also see a similar pattern in the semi-oracles where knowing the emission density and nonsequential CCPs is more beneficial than knowing the underlying GMM. However, the difference between these is not as stark, proportionally speaking. One possible explanation for part of this is that, for all values of \( M \), the entropy rate\(^1\) as shown in table 11 of the Discrete Beta model is higher, meaning the sequence of states is less predictable on average and thus provides less information to us. We also notice that performance of the two semi-oracles appears more sensitive to increasing noise and increasing \( M \).

\(^1\)The entropy rate for a Markov chain is given as \( H(Y_t) = - \sum_{ij} \mu_i A_{ij} \log A_{ij} \) where \( \mu \) is the vector of stationary probabilities and \( A \) is the transition matrix. For more details see Cover and Thomas [16].
Figure 24: Performance for the Discrete Beta holding time GMM as a function of $\sigma$. Each panel is a different value for $M$. Fit Type indicated whether we know the nonsequential conditional class probabilities $P(Y_t | X_t)$ or the GMM transition structure on $(Y_t)$.
Figure 25: Performance for the Beta Negative Binomial with Geometric tail holding time GMM as a function of $\sigma$. Each panel is a different value for $M$. Fit Type indicated whether we know the nonsequential conditional class probabilities $P(Y_t \mid X_t)$ or the GMM transition structure on $(Y_t)$.
5.1.2. Attempting to Resolve the Discrepancy

It is unexpected that additional information would yield worse predictions but this is what we observe in the preceding simulations. So naturally, we would like to ask why performance decreases when we use partially true and partially estimated models as opposed to fully estimated models. As we show in this section, at least part of the discrepancy comes from an additional piece of information that we are not using correctly; the marginal class distribution $P(Y_t)$.

To see the relevance of the marginal distribution to our simulations, consider that if we know the transition matrix of a Markov chain, then we know the stationary distribution $\mu$, which is the marginal distribution. However, if we fit a model to estimate $\hat{P}(Y_t = i \mid X_t)$ and compute $\hat{p}_i$ of $p_i = P(Y_t = i)$ from the data by looking only at the marginal counts it may not be consistent with $\mu$. Further, if the $Y$ generating process is sufficiently complex relative to the sample size, the marginal distribution in the training set can differ from that in the test set. This may not only affect our estimate of $p_i$ but also of the conditional class probabilities, since it will yield a different empirical distribution on $X$. If we were working with generative models, the problem would be even more apparent; even if $f(X \mid Y)$ is known and invariant from sample to sample, the joint density is directly affected by changes in the marginal distribution of either $X$ or $Y$. For i.i.d. data, none of this would not be a concern, as the marginal distribution would vary little from simulation to simulation, but we will see shortly that it makes a difference in cases like the preceding.

Figures 26 and 27 show the variability in observed marginal class probabilities across simulated datasets using the same sequential models and sample sizes we saw in the preceding
simulations where there are three possible labels. Each observation is represented by a point on a ternary plot, which represents three dimensional points that must sum to one on the two dimensional simplex with the labeled lines opposite a labeled corner of the triangle representing points with an equal probability of that corner’s label. The marginal class probabilities naturally vary much less for larger training sets but more than we would expect for i.i.d data. We also note that increased training set size does not seem to help the semi-oracles, proportionally speaking, so variability in marginal distributions between simulations do not tell the full story here.

Figure 26: Ternary plots of the observed and expected marginal distributions when $T = 1000$ for $M = 3, 10$. Points along lines running parallel to a side opposite a labeled vertex have the same probability for that class.
Figure 27: Ternary plots of the observed and expected marginal distributions when $T = 10000$ for $M = 3, 10$. Points along lines running parallel to a side opposite a labeled vertex have the same probability for that class.
5.2. Using the Marginal Distribution

Given that we believe our knowledge or estimation of the marginal class probabilities is relevant to improving our classifiers’ performance, what is the appropriate way to use them? Depending on which piece we know we will want to use them differently. In the context of the preceding simulations, there are four “things” that we could know. First, we might know the Markov model, which also implies knowledge of the marginal class probabilities. Second, we might only know the true emission distribution \( P(X_t | Y_t) \). Third, we could know the marginal class probabilities, \( P(Y) \) and nothing else. Finally, we could know both the emission distribution and the marginal class probabilities, but not the process that generated \((Y_t)\), which also implies a knowledge of the true conditional class probabilities. Given the data’s generating process and assuming we can only know pieces of that, this last case is the only situation in which we have access to the true conditional class probabilities. Notice that this is an expanded view of what could be known in comparison to the original view from the preceding simulations. It also clarifies the distinction between knowing the emission distribution (a piece of the model) and truly knowing the conditional class probabilities.

With this new view, we now detail each of the above scenarios, describing what exactly we know and potential methods to incorporate the additional information in the context of our simulations. Following this, we present an extended set of simulations to assess whether this view and methodology yield a resolution to the previously unexpected results.

5.2.1. When Only Marginal Class Probabilities are Known

If we know only the true \( p_i = P(Y_t = i) \) then there are two places this information could be used. First, even though we used logistic regression, which is a purely discriminative classifier, we will see that knowledge of the marginals can nevertheless affect output on a test set. Further, we have reason to believe this is generally true for most classifiers. The general method used in the next set of simulations involves rescaling by a ratio of the true and sample marginal distribution and is discussed in depth in section 5.4.
Second, we would want to change how we handle the estimation of the Markov process on \((Y_t)\). Ideally, we would like to fit the best model constrained to have a given stationary distribution. However, we have not seen any work on this in the literature for even simple Markov chains, let alone more complicated models like GMMs and such problems lie beyond the scope of the current work. Therefore we do not discuss this case further or address it directly in the following simulations. However, there is a relatively simple way we could consider incorporating the marginals. When smoothing nonsequential probabilities with the discriminative Forward-Backward algorithm, recall that we use the term \(\hat{P}(Y_t=i|X_t)\) in each update step. Since the denominator is the marginal probability of observing class \(i\), we replace it with \(p_i\), the true marginal probability of class \(i\).

5.2.2. When The Emission Distribution Is Known

In this condition there are also two sub-scenarios; we could either estimate or know the marginal class distributions. The first scenario would result in us estimating the nonsequential CCPs as \(P(X_t|Y_t)^{\hat{P}(Y_t)}\). This may be worth looking into, but we do not discuss it further at present. In the second, we actually know the true CCPs and could set our initial, nonsequential estimates to the true \(P(Y_t|X_t)\). Were there no sequential model and hence no smoothing to be done, there would be no more to do since this would be the truth.

When we know the true nonsequential conditional class probabilities, however, we need to be careful at the smoothing step. If we just used the emission distribution in the smoothing step, the nonsequential CCPs wouldn’t even be relevant. But since sequential learning uses discriminative models, the marginal class probabilities show up in this step. Treating the nonsequential CCPs as known but the marginal probabilities as unknown and using an estimate in the denominator is thus an inconsistent use of available information.

Formally, we use \(P(X_t|Y_i=i)\) or an estimate thereof in the generative version of Forward-Backward but recall that in the generative case we use the fact that \(P(X_t|Y_t=i) \propto \frac{P(Y_i=i|X_t)}{P(Y_t=i)}\) replacing the terms by estimates when we do not know the truth. So, if an oracle
gives us only the true conditional class probabilities but not the true marginal distribution, we end up and incorrect emission distribution proxy of \( \frac{P(Y_t = i \mid X_t)}{P(Y_t = i)} \) because the correct marginals were used in creating the CCPs. The general principle seems to be that we want to pair truth with truth and estimates with estimates.

5.2.3. When the state transitions are known

The final and most interesting condition is when we know \( A \) and the holding time parameters — in other words the complete Markov model. With this information, the marginal class distribution is included for free in the form of the process’s stationary distribution. However, the way in which we use this information is less direct. A discriminative classifier that directly estimates \( \hat{P}(Y_t = i \mid X_t) \) as was used in the preceding simulations makes no explicit mention of \( p_i \). However, differences in marginal distribution from sample to sample may still affect this estimate so we would like a way to incorporate this extra knowledge into any estimates since it may improve out of sample performance.

The method we use, while less straightforward, is still relatively simple. First, recall that \( \hat{P}(Y_t = i \mid X_t) \propto \hat{f}(X_t \mid Y_t = i) \hat{p}_i \). However, we don’t explicitly have the emission distribution which would let us just replace \( \hat{p}_i \) by \( p_i \) and \( \hat{f} \) by \( f \) to exactly reconstruct the conditional class probabilities. Besides, this would mean we know the whole model anyway. Instead, starting from any estimate \( \hat{P}(Y_t = i \mid X_t) \) of the CCPs we let \( \hat{P}^*(Y_t = i \mid X_t) = \hat{P}(Y_t = i \mid X_t) \frac{p_i}{\hat{p}_i} \). Notice that when \( P^* \) is used to smooth probabilities, since we know the true \( p_i \) this correction yields \( \frac{\hat{P}(Y_t = i \mid X_t)}{\hat{p}_i} \). This is in contrast to the preceding where we ended up with truth over truth.

Finally, we point out that this type of correction to class probability estimates requires further investigation and we discuss this in much greater depth in section 5.4.
5.3. Results with Modified Classifiers

Now we examine the results of our marginal-augmented methods alongside the originals in figures 28 and 29. The plots were made using the same methods as figures 24 and 25 with the addition of “Known Time Series + Marginal” and “Known Non-Sequential + Marginal” which respectively use the true marginal distributions to modify the estimated CCPs and smoothing step as previously described. While using the correct marginal distribution does not uniformly improve performance, it does in most cases and improves performance regardless of which other piece of the model we know.

When the sequential model has discrete beta holding times (figure 28), we first notice that regardless of sample size, the model using the true CCPs without correction performs best when $M \leq 7$. However, knowing and using the marginals has a stabilizing effect on model performance, with it behaving similarly regardless of $M$. For $T = 1000$ we find the corrected version performing best at $M > 7$, which makes sense in light of our expectation that the marginal distribution in any sample will diverge from its average more as the sequential model becomes more complex.

On the other hand, when $T = 10000$ we see a similar general pattern except that for $M > 7$ the corrected model where we know the GMM performs best. At first this is somewhat surprising, as the unmodified version of this model does the worst, but on second though it is understandable. When the sample size is large, even if the marginal distributions vary substantially, the relationship between $Y$ and $X$ is simple enough that the logistic regression can capture it easily, so a corrected version of this classifier should be very similar to knowing the truth. Another reason for this observation, and the fact that this model appears to have the most stable errors as a function of $M$ is that getting a good model for the time series becomes more difficult at a given $T$ as $M$ increases. Since it already has the “hard” part of the model and the easy part is corrected so as to be consistent across simulations we expect little variation between scenarios.
Figure 28: Performance for the Discrete Beta holding time GMM as a function of $\sigma$. Each panel is a different value for $M$. Fit Type indicated whether we know the nonsequential conditional class probabilities $P(Y_t|X_t)$ or the GMM transition structure on $(Y_t)$. The ‘+ Marginal’ designation is when we use the true marginal class distribution with the method as opposed to the estimate distribution.
When the sequential model is more complex (figure 29) we see that adding in marginal distribution information again helps a lot but the qualitative nature of the improvement is not identical to the first simulation. Also, the relative performance of the different methods is more consistent for a given value of $T$, with at least one of the corrected models always beating the all estimates one. Referring back to table [II], we notice that even the simplest sequential model in this set of simulations has an entropy rate about as low as the point where correction yields consistent improvement in the discrete beta case. While this surely does not tell the entire story, it does indicate the effect that Markov model complexity can have on the outcome here.

When $T = 1000$, the corrected model where we know the true CCPs consistently outperforms the all estimates one. Unless the noise level is low, the corrected known time series model still underperforms the all estimates model, but when $\sigma$ is small but not close to zero, we can see that it is actually winning by a small margin. Knowing the true CCPs along with the marginals, however, consistently beats using estimates of both the time series and CCP. Further, its performance is stable and does not increase substantially with noise. This highlights the importance of starting with good nonsequential estimates and that a worse estimate of the sequential part is less detrimental — similar to what we saw with low order Markov chains performing surprisingly closely to results obtained from VLMCs.

The difference between the two marginal-corrected method practically disappears when $T = 10000$. This is consistent with the previous set of simulations where more data to estimate the nonsequential part of the model yielded greater improvement than more data to estimate the time series. However, we note a difference here that this method no longer wins uniformly across all values of $\sigma$. It consistently performs the best not only when $\sigma$ is 1/2 as before, but also at $\sigma = 1$. But it still performs worse than using all estimates as $\sigma$ increases and the $M$ is large. This phenomenon is curious as it points out a dependence between the sequential and nonsequential parts of the model. If the sequential part is complex, it has a larger effect on the true probabilities, so if we know that and start with
worse estimates of the nonsequential part, the strong influence of the sequential part may compound this problem.

While using the true marginal class probabilities certainly helps and partially resolves the original counterintuitive results, a few elements of these simulations remain to be studied and understood. For instance, in both sets of simulations it appears that increased training sample size helps more on an absolute basis when we are estimating the nonsequential $P(Y_t | X_t)$ as opposed to estimating the underlying Markov process, even going so far as for the relative rankings of the methods to change. While we described some intuition for this phenomenon, it will be interesting to quantify it and understand what controls it directly. We also would like to better understand what affects how large an impact knowing the marginals will have on a method and why the relative effect on different methods changes as sample size increases.

5.4. Classification with Known Marginals

Using the preceding simulation study as motivation, we now ask the question: “Given the marginal class distribution $P(Y)$ how can we incorporate this information into an estimate $\hat{P}(Y | X)$?” Under classical assumptions where the $(X,Y)$ are all i.i.d pairs the question is less relevant, as even with moderate sample sizes the training and test set marginal probabilities will be close.

However, it may still be relevant, because if one class is rare, fitting on a representative dataset can lead to it being underpredicted [29] while minimally affecting the overall performance of the classifier. This issue commonly manifests itself in instances where one class is rare are and the training data is selected such that the rare class is overrepresented and would thus be overpredicted out of sample. Generally speaking, the idea that our training and test sets can come from distributions is known as sample selection bias [30, 66] and captures a number of different ways in which they can differ.
Figure 29: Performance for the Beta Negative binomial with Geometric tail holding time GMM as a function of $\sigma$. Each panel is a different value for $M$. Fit Type indicated whether we know the nonsequential conditional class probabilities $P(Y_t | X_t)$ or the GMM transition structure on $(Y_t)$. The ‘+ Marginal’ designation is when we use the true marginal class distribution with the method as opposed to the estimate distribution.
5.4.1. Similar Problems

Work has been done on what is variously known as transfer learning [46] and covariate shift [7] with a focus on what happens when the marginal distribution of $X$, rather than $Y$ differs between datasets or a focus on changes in the joint distribution of $X$ and $Y$.

In addition to requiring the more challenging task of knowing or estimating possibly complex conditional densities, these methods highlight an important philosophical difference with ours. Work such as Bickel et al. [7], Gretton et al. [27] and Shimodaira [56] makes the assumption that $P(Y \mid X)$ does not change whereas we will assume that $P(X \mid Y)$ is what is constant. This speaks to how the data generating process is viewed; in the first case, we can think of a two processes that creates a scattering of $X$’s but that no matter what made them, as long as an $X$ lands in the same place, it has the same chance of a given type of $Y$ popping out of it. The latter assumption views the data as $X$ covariates being “emitted” from a data point as a function of what type of $Y$ it happens to be. To see that these are not the same, consider allowing $P(Y)$ to change but fixing $P(X \mid Y)$; the marginal distribution of $X$ will, of course, still change, but it is restricted in that it is always going to be a mixture distribution with the same components but different weights, whereas the “fix $P(Y \mid X)$” case allows the marginal distribution of $X$ to change arbitrarily.

The last we will say about this, since it is beyond the scope of this work, is to mention a result that ties in with our work here. Shimodaira [56] and Bickel et al. [7] show that if we have a generative model with two joint distributions $(x, y) \sim \lambda$ and $(x, y) \sim \theta$, a classifier, $f$, and a loss function $\ell$, then the expected losses are related by:

$$E_{(x, y) \sim \theta}(\ell(f(x), y)) = E_{(x, y) \sim \lambda}\left(\frac{p(x \mid \theta)}{p(x \mid \lambda)} \ell(f(x), y)\right).$$

This idea is reminiscent of importance sampling in that we can re-weight points by the likelihood ratio given by the two distributions to get an expected loss under a different distribution without having to refit. Notice that only the density of $X$ matters in this case.
since that is what changes and the conditional class probabilities are assumed not to change. We will see the idea of ratio reweighting rear its head again shortly. However, since our framework assumes changing marginal class probabilities and invariant emission densities, it will show up with class probabilities instead of covariate densities.

5.4.2. If I had a prior

If we were either given the conditional distributions of covariates \( f_Y(X) \) or estimated them and were asked to compute conditional class probabilities \( P(Y|X) \) we would need to know \( P(Y) \) to reconstruct them. And if \( P(Y) \) varied between populations or between a training and test set, this would not be an issue as long as \( f_Y(X) \) were the same. This is exactly what happens when using Linear Discriminant Analysis, as it is assuming a parametric (Normal, in fact) \( f_Y(X) \), estimating the parameters and then uses whatever prior on \( Y \) you choose.

Most classification methods, however, estimate \( P(Y|X) \) directly and \( P(Y) \) is left implicit. We can, however, use this information to adjust our conditional estimates as discussed in Elkan [19], Weiss and Provost [62], and King and Zeng [33]. Expressed succinctly, the corrected probability estimates are

\[
\hat{P}(Y = i | X) = \frac{\hat{P}(Y = i | X) \frac{p_i}{\hat{p}_i}}{\sum_{j=1}^{k} \hat{P}(Y = i | X) \frac{p_j}{\hat{p}_j}} \tag{*}
\]

where \( \hat{p}_i \) is the marginal frequency of class \( i \) in the training sample (or population) and \( p_i \) is the known probability of class \( i \) in the test population. To justify this, we need one assumption: that \( \hat{f}_i(X) = \hat{f}_i(X) \) where the two \( f \)s are the density of the covariates in and out of sample. Without this assumption it would mean that for any given class, the covariate distributions would be different, thus changing the relationship between labels and covariates.
Equivalence to Other Methods

Equation \( \text{[\ref{eq:equivalence}] \url{https://doi.org/10.7554/eLife.19338.008}} \) appears in the appendix to King and Zeng \[33\] who discuss it as an asymptotic estimate to the true conditional class probability. However, it is more than that as it is equivalent to an expression given in Elkan \[19\] for two-class problems which relates two different true conditional class probabilities between two populations. To see this equivalence make the following manipulations

\[
\frac{\hat{P}(Y = 1 \mid X) \frac{p_1}{\hat{p}_1}}{\hat{P}(Y = 1 \mid X) \frac{p_1}{\hat{p}_1} + \hat{P}(Y = 0 \mid X) \frac{p_0}{\hat{p}_0}} = \frac{\hat{P}(Y = 1 \mid X) p_1 \hat{p}_0}{\hat{P}(Y = 1 \mid X) p_1 \hat{p}_0 + \hat{P}(Y = 0 \mid X) p_0 \hat{p}_1} \\
= \frac{\hat{P}(1 \mid X) p_1 (1 - \hat{p}_1)}{\hat{P}(1 \mid X) p_1 (1 - \hat{p}_1) + \left( 1 - \hat{P}(1 \mid X) \right) (1 - p_1) \hat{p}_1} \\
= p_1 \frac{\hat{P}(1 \mid X) (1 - \hat{p}_1)}{\hat{P}(1 \mid X) p_1 + \hat{p}_1 - p_1 \hat{p}_1 - \hat{p}_1 \hat{P}(1 \mid X)}
\]

which give the formula found in Elkan \[19\]. While the correspondence between the two adjustments is simple, it helps provide intuition as it is clear and intuitive.

For logistic regression, a popular method of handling imbalanced data is known as prior correction and is commonly used in case-control studies where the training data are often taken so that the number of instances of, say, a disease are roughly balanced with those of healthy patients \[47, 1\]. In this case, the emphasis is not so much on the fact that the marginal distribution of classes is different but that rare events are oversampled \[33\] and would thus be overpredicted as opposed to underpredicted out of sample. This is also done because such data can be time consuming and expensive to collect and it is important to not risk having few or even no instances of one class in the training data. In our discussion of this method, we revert to considering the binary case where \( Y \) is just 0 or 1, but the results generalize easily.

If \( p \) is the true proportion \( P(Y = 1) \) of the population and \( \hat{p} \) is the proportion of 1’s in our sample, then setting \( \hat{\alpha}^* = \hat{\alpha} + \log \frac{p}{1 - p} - \log \frac{\hat{p}}{1 - \hat{p}} \), as given in Fienberg \[21\], where \( \hat{\alpha} \) is the
intercept term estimated from our biased sample, will correct our estimate of \( P(Y = 1 \mid X) \).
The intuition here is that, if we assume the relationship between the outcome and covariates is the same regardless of the marginal probabilities, then the only thing we can change to affect the estimated marginals is the intercept and doing so will not change this relationship.

We now show the derivation of this formula as given in [21] and begin by defining the latent variable

\[
Z = \begin{cases} 
1 & \text{if subject is included in sample} \\
0 & \text{otherwise} 
\end{cases}
\]

Let \( \pi_1 = P(Z = 1 \mid Y = 1) \) and \( \pi_0 = P(Z = 1 \mid Y = 0) \). Note that \( Z \) is independent of \( X \) since the inclusion criterion is only based on the values of \( Y \). Using this, we get

\[
P(Y = 1 \mid Z = 1, X) = \frac{P(Z = 1 \mid Y = 1, X) P(Y = 1 \mid X)}{\pi_1 P(Y = 1 \mid X) + \pi_0 P(Y = 0 \mid X)}
\]

and

\[
P(Y = 0 \mid Z = 1, X) = \frac{\pi_0 P(Y = 0 \mid X)}{\pi_1 P(Y = 1 \mid X) + \pi_0 P(Y = 0 \mid X)}.
\]

So, the likelihood ratio conditional on being in the sample and unconditional differs only by a constant factor because

\[
\frac{P(Y = 1 \mid Z = 1, X)}{P(Y = 0 \mid Z = 1, X)} = \frac{\pi_1 P(Y = 1 \mid X)}{\pi_0 P(Y = 0 \mid X)} \tag{5.1}
\]

To get an actual value for the correction factor, note that

\[
\frac{\pi_1}{\pi_0} = \frac{P(Z = 1 \mid Y = 1)}{P(Z = 1 \mid Y = 0)} = \frac{P(Y = 1 \mid Z = 1, X) P(Z = 1)}{P(Y = 1 \mid Z = 0, X) P(Z = 1)} = \frac{P(Y = 1 \mid Z = 1) P(Y = 0)}{P(Y = 0 \mid Z = 1) P(Y = 1)} = \frac{\hat{p}(1-p)}{(1-\hat{p})p}
\]

Since we are estimating the left hand side of (5.4.2), we get the stated correction factor.

More generally, when the possible values of \( Y \) are 1, \ldots, \( K \), multinomial logistic regression
gives us that the likelihood ratio of class $i$ with class $K$ is

$$\frac{P(Y = i \mid X)}{P(Y = K \mid X)} = e^{\alpha_i + \beta_i^T X}$$

Defining $\pi_i = P(Z = 1 \mid Y = i)$ we can get analogous corrections to the two class case.

Of note here is that when we do this correction, $Y$ is fixed, in a sense, since we choose subjects based on their value of $Y$ and only observe $X$ after the fact. This makes $X$ our random quantity, in contrast to a study where we would fix $X$ and then observe the outcome $Y$.

**Proposition 5.** The prior correction intercept adjustment is equivalent to scaling the estimate, $P(Y = i \mid X, Z = 1)$ of by $\frac{\pi_i}{\hat{p}_i}$, renormalizing as appropriate.

**Proof.** Let $c$ be the sum of the rescaled probabilities. In the binary case, the likelihood ratio of the new probabilities is

$$\frac{\frac{1}{c} \hat{p} P(Y = 1 \mid X, Z = 1)}{\frac{1}{c} \left(1 - \hat{p}\right) P(Y = 0 \mid X, Z = 1)} = \frac{P(Y = 1 \mid X, Z = 1) p(1 - \hat{p})}{P(Y = 0 \mid X, Z = 1) \hat{p}(1 - p)}.$$  

Trivially, this also holds for $K$ outcome levels. \hfill \Box

**How Much Does Correction Help?**

It turns out we can actually quantify the improvement that rescaling conditional class probabilities gives us. Overloading notation a bit, let $\hat{p}(X)$ be the conditional class probability vector given by $\hat{P}$ for a random new instance with covariates $X$ and let $\hat{p}_Y(X) = \hat{P}(Y \mid X)$. Then define $\tilde{p}(X)$ and $\tilde{p}_Y(X)$ correspondingly. Also let $\hat{f}$ be the implied density of $X$ under $\hat{p}$ and let $f$ be the density of $X$ under $p$. With these definitions we can proceed.

**Proposition 6.** Under log loss, the improvement from using $\tilde{P}(Y \mid X)$ over $\hat{P}(Y \mid X)$ is

$$E(-\log (\tilde{p}_Y(X))) = E(-\log (\hat{p}_Y(X))) - \left(\text{KL}(p || \hat{p}) - \text{KL}(f || \hat{f})\right)$$
where the expectations are taken over \((X,Y)\) using the true \(p\) and \(f\) and \(D_{KL}\) is, of course, the Kullback-Leibler divergence.

Now there are two things to do. First, some algebraic manipulation to get the equality. And second, to show that the \(D_{KL}(p||\hat{p}) - D_{KL}(f||\hat{f})\) \(\geq 0\) which makes the result meaningful; that’s the fun part. On to the algebra.

**Proof.** First, note that \(P(X,Y = i) = f_i(X)p_i\), so the expected loss can be expressed as

\[
E\left(-\log(\hat{p}_Y(X))\right) = -\int \sum_i \log(\hat{p}_i(x)) f_i(x)p_i\,dx
\]

\[
= \int \sum_i \log\left(\frac{\hat{p}_i(x)p_i}{\sum_j \hat{p}_j(x)p_j}\right) f_i(x)p_i\,dx
\]

\[
= \int \sum_i f_i(x)p_i \left[ \log(\hat{p}_i(x)) + \log\left(\frac{p_i}{\hat{p}_i}\right) - \log\left(\sum_j \hat{p}_j(x)\frac{p_j}{\hat{p}_j}\right) \right] \,dx
\]

\[
= E\left(-\log(\hat{p}_Y(X))\right) + \int \sum_i f_i(x)p_i \left[ \log\left(\frac{p_i}{\hat{p}_i}\right) - \log\left(\sum_j \hat{p}_j(x)\frac{p_j}{\hat{p}_j}\right) \right] \,dx
\]

Now that we’ve expressed the loss over the corrected classifier in terms of the uncorrected one, we need to handle the second term on the last line. Going inside the sum and integral, we take it one term at a time for the first inner term, exchange the sum and integral and note that \(f_i(x)\) is a proper density to get

\[
\int \sum_i f_i(x)p_i \log\left(\frac{p_i}{\hat{p}_i}\right) \,dx = \sum_i \int f_i(x)p_i \log\left(\frac{p_i}{\hat{p}_i}\right) \,dx = \sum_i p_i \log\left(\frac{p_i}{\hat{p}_i}\right) = D_{KL}(p||\hat{p})
\]

The second inner term is trickier. We must first note that \(\frac{\hat{p}_j(x)}{\hat{p}_j} = \frac{f_j(x)}{f(x)}\), recalling that \(f_j(x)\) is the same no matter what distribution we have over \(Y\). Then, we can express \(f_j(x)p_j\) as \(P(X,Y = j)\). Therefore, the sum inside the log is just \(\sum_j \frac{P(X,Y = j)}{f(x)} = \frac{f(x)}{f(x)}\). After all this,
we are left with

\[- \int \sum_i f_i(x)p_i \log \left( \frac{f(x)}{\hat{f}(x)} \right) dx = - \int \log \left( \frac{f(x)}{\hat{f}(x)} \right) \sum_i f_i(x)p_i dx \]

\[= - \int \log \left( \frac{f(x)}{\hat{f}(x)} \right) f(x) dx \]

\[= -D_{KL}(f||\hat{f}) \]

thus proving the result. \hfill \Box

With the preceding result in hand we provide a lemma to make it useful by showing that the given equivalence actually implies an improvement in expected loss from adjusting the probability estimates.

**Lemma 1.** $D_{KL}(p||\tilde{p}) \geq D_{KL}(f||\hat{f})$.

**Proof.** We will show this result by manipulating $\frac{f(x)}{\hat{f}(x)}$ to give us what we want. To do this, first note that plain old Bayes’ rule tells us that

\[p_i(x) = \frac{f_i(x)p_i}{f(x)} \quad \hat{p}_i(x) = \frac{f_i(x)\tilde{p}_i}{\hat{f}(x)}.\]

Taking ratios here and solving for the desired quantity gives

\[\frac{f(x)}{\hat{f}(x)} = \frac{\hat{p}_i(x)p_i}{p_i(x)\tilde{p}_i}\]

Notice that this is true for any $i$ we choose. Next, remember that we can write $f(x) =
\[ \sum_i f_i(x)p_i \text{ so we can express the } D_{KL} \left( f \| \hat{f} \right) \text{ as} \]

\[ D_{KL} \left( f \| \hat{f} \right) = \int \sum_i f_i(x)p_i \log \left( \frac{f(x)}{\hat{f}(x)} \right) dx \]

\[ = \int \sum_i f_i(x)p_i \log \left( \frac{\hat{p}_i(x)p_i}{p_i(x)\hat{p}_i} \right) dx \]

\[ = \int \sum_i f_i(x)p_i \left[ \log \left( \frac{\hat{p}_i(x)}{p_i(x)} \right) + \log \left( \frac{p_i}{\hat{p}_i} \right) \right] dx \]

\[ = \int \sum_i f_i(x)p_i \log \left( \frac{\hat{p}_i(x)}{p_i(x)} \right) dx + \int \sum_i f_i(x)p_i \log \left( \frac{p_i}{\hat{p}_i} \right) dx. \]

The second term is just \( D_{KL} (p \| \hat{p}) \) because

\[ \int \sum_i f_i(x)p_i \log \left( \frac{p_i}{\hat{p}_i} \right) dx = \sum_i p_i \log \left( \frac{p_i}{\hat{p}_i} \right) \int f_i(x) dx = \sum_i p_i \log \left( \frac{p_i}{\hat{p}_i} \right). \]

To handle the first term, recall that \( f_i(x)p_i = p_i(x)f(x) \) so we have

\[ \int \sum_i f_i(x)p_i \log \left( \frac{\hat{p}_i(x)}{p_i(x)} \right) dx = \int \sum_i p_i(x)f(x) \log \left( \frac{\hat{p}_i(x)}{p_i(x)} \right) dx \]

\[ = \int f(x) \sum_i p_i(x) \log \left( \frac{\hat{p}_i(x)}{p_i(x)} \right) dx \]

\[ = - \int f(x) D_{KL} (p(X) \| \hat{p}(X)) dx \]

The integral on the last line is always nonnegative because \( D_{KL} \geq 0 \) and we are integrating the product of nonnegative functions. Therefore, the last line is negative and the result is proved. \( \square \)

What the above result tells us is not only that we will do better under log loss with corrected probabilities but by exactly how much. To understand this result better, it helps to think about extreme cases when the difference is as large or as small as possible. These occur when the classes are either perfectly separable or the covariates give no information at all. In the former case, we have \( D_{KL} (p \| \hat{p}) = D_{KL} \left( f \| \hat{f} \right) \) because knowing \( X \) uniquely identifies the
class label and prior class probabilities should have no bearing on our classification. On the other hand if \( f_i(X) \) is the same for all \( i \), then the class probabilities are all we have to go on and \( D_{KL}(f||\hat{f}) = 0 \). In this case, any differences in classification will be entirely due to differences in the base rate so the distance between the training and test class probabilities is all that matters.

Unfortunately, the above proposition one of those things that is nice to know but that we usually won’t be able to compute explicitly. Intuitively, it is the average of the divergence between corrected and uncorrected probability estimates over the possible out of sample values of \( X \). In practice, the marginal density of \( X \) is often complicated and difficult to estimate. But it’s still nice to know!

5.4.3. Simulations

Here we attempt to better understand the effects adjusting probability estimates can have on classifier output when marginal probabilities vary from sample to sample. To that end, we generate the sequence of labels \( (Y_n) \) from a VLMC generated by running the VLMC fitting algorithm on the sleep stages of mice from the same data we use in chapter 5. Such a structure on such data was chosen because the resulting trees have a complex structure of a few hundred leaves and thus can easily produce substantial variability in the sequences generated from simulation to simulation. In order to better understand applications of the method beyond theory we consider misclassification loss as well as log loss and use logistic regression as well as random forests as base classifiers. Random forests are especially interesting in this regard because they are known to often do well with respect to misclassification but produce poor conditional class probabilities in the absence of calibration [41, 8]. The distinction between these two methods will help understand how much or little correction can help depending on the quality of the original conditional class probability estimates.

In addition to considering multiple loss functions and classifiers, the simulations are designed
to illuminate performance gains of marginal probability correction as a function of how much test sets vary from the marginal class probabilities and as a function of how noisy the covariates are. To this end the covariates were generated so that \( X|Y \sim \mathcal{N}(\mu_Y, \sigma I) \) where \( I \) is the identity matrix, \( \sigma \in \{0.02, 0.1, 0.5, 1, 4 \} \) and \( \mu_N = (0, 0) \), \( \mu_W = (1, 0.5) \), \( \mu_R = (0.2, -0.1) \). For each level of \( \sigma \) we construct 2000 training sets of size 10000 and for each training set, we simulate 400 test sets of size 3000. While a reasonably large test set size, this still ensures substantial variability in the marginal class probabilities in the test sets as can be seen in figure 30, which provides a ternary plot showing a number of sample marginal class distributions.

The reason for using many test sets per training set is that in the end we want to assess performance as a function of the training set distribution on unknown test sets. There is, naturally, variability in the performance of both our base classifiers and corrected output, and the effectiveness of correction may depend on which two we are looking yet. So, we need to ensure that our assessment is based on how much correction helps in aggregate for a given test set.

![Distribution of Class Proportions, n=3000](image_url)

Figure 30: Ternary plot of the true marginal class probabilities in the simulation along with those of 100 sample paths of length 3000.
Results

To assess the performance of the method as a function of test set variability, the $x$ axis in our comparison plots are the proportion of the state $w$ in the test set, put into 100 bins. For each bin, we consider the sum and difference of log loss (figures 31 and 32) for corrected and uncorrected conditional class probabilities as well as the sum and difference of the misclassification loss (figures 34 and 35). We also provide plots of the raw loss for each method to get another view of how the methods compare (figures 33, 36).

![Log Loss Difference with Logistic Regression](image)

![Log Loss Difference with Random Forests](image)

Figure 31: Difference in classifier performance under log loss. Negative numbers indicate corrected loss is lower than uncorrected. Notice that with a few points close to 0 in the very low noise case, which is almost certainly a numerical artifact, the corrected estimates perform better.

First, let’s consider the effect that marginal probability correction has on log loss in the simulations. Consistent with the theory, and with a few seemingly numerical artifacts, rescaling performs universally better. It is interesting that it does so even with random forests which often produce notably poor conditional class probabilities. Further, note that while the difference in log loss between methods grows faster as a function of $P(w)$ for noisier data the same does not appear to be true for the ratio, with the exception of low noise simulations where the difference is effectively 0. This is also apparent from figure...
Figure 32: Ratio in classifier performance under log loss. Numbers less than 1 indicate corrected loss is lower than uncorrected. Notice that with a few points close to 1 in the very low noise case, which is almost certainly a numerical artifact, the corrected estimates perform better.

Figure 33: Log loss for corrected and uncorrected probabilities using logistic regression and random forests. Notice that random forests is uniformly worse but that correction still helps and that the loss as a function of $P(w)$ is much less symmetric.
where we see that asymmetry in the loss for random forests is masked by a roughly
symmetric difference and ratio for the two methods. We also point out that the flat-looking
loss curves for corrected probabilities in figure 33 do not seem to come from any sort of
mistake in the simulation, as inspection of the data reveals some variation in the numbers.
So it does seem that rescaling can, at least in some cases allow much more consistent
performance across test sets where the marginal distribution of class labels can vary.

Finally, we point out that marginal correction improves both methods by similar amounts,
which can be large in extreme cases, but still moderate and meaningful in cases where
the sample marginal distribution is not so far from the true marginals. In light of this, it
is perhaps reasonable to assume that, at least for classifiers that do not tend to produce
unreasonably extreme probability estimates, the gains from rescaling would be similar.

Figure 34: Difference in classifier performance under misclassification loss. Negative num-
bers indicate corrected loss is lower than uncorrected. Several points close to 0 may be a
numerical artifact, or, in the case of random forests with high noise, be due to the overall
poor performance of the base classifier.

Next, consider misclassification loss where we have no theory yet to back us up. However,
just like with log loss, we see a relatively consistent performance increase from rescaling
conditional class probabilities. On the other hand, while the relationship between test set
Figure 35: Ratio in classifier performance under misclassification loss. Numbers less than 1 indicate corrected loss is lower than uncorrected. The missing plots are due to 0 loss in the very low noise case.

Figure 36: Misclassification loss for corrected and uncorrected probabilities using logistic regression and random forests. Notice that random forests is uniformly worse but that correction still helps and that the loss as a function of $P(w)$ is more symmetric than when considering log loss.
marginals and performance looks similar for low noise levels, for the highest amount of noise we see what appears to be a qualitative difference. For random forests, it seems that the benefits of knowing marginal probabilities is almost nonexistent for highly noisy covariates, whereas logistic regression retains the benefits, albeit with them leveling off more quickly than in the case of log loss. This indicates that improvements from correction are contingent on the quality of the initial estimates.

Finally, we note that while the simulations study a particular type of data with particular classifiers, we have strong reason to expect similar results in many other scenarios. Marginal rescaling operates only on the probability estimates and the size of the benefit, at least on log loss, is a function of the marginal and emission distributions that is not directly influenced by the dimensionality of the data in either the number of covariates or class labels. This provides ample reason to believe our results are generally applicable for diverse data.
Chapter 6

Conclusion and Future Work

Our work thus far has achieved two main goals. First, we have increased our understanding of the sequential learning framework both theoretically and empirically and extended it to using Variable Length Markov Chains, which allow it to be used with fewer assumptions on a larger more general set of problems. In addition to this extension, there is now a better understanding of the workings of the method, both through proofs which provide intuition for its empirical behavior by looking at edge cases, and through simulations which demonstrate both strengths and weaknesses of the method. In these simulations we note that sequential learning with VLMCs can provide a large benefit over nonsequential methods, but that this cannot overcome poor initial estimates.

We have also applied this method to mouse sleep data and shown that it performs at least as well and often better than a method using a sequential model specifically chosen for this data. This level of generality can, if a user desires, turn sequential learning into a “black box” by building both the sequential and nonsequential parts of the model with little to no human intervention.

Second, inspired by observations made studying sequential learning, we have developed a general method to improve the probability estimates produced by classifiers when additional
information about the marginal distribution is known. This method of marginal probability reweighting has a simple, intuitive form and has been shown to generalize extant methods used in similar problems. A related form of this method also was shown to resolve seemingly incongruous results from the simulations that inspired it. Further, we have not only demonstrated the value of the method of marginal probability reweighting in simulation but we have provided a proof of the magnitude of improvement under log loss for any initial probability estimates.

6.1. Future Work

We have noticed that, using sequential learning, all probability estimates tend to be pushed toward 0 and 1 and yield uncalibrated estimates, even with fairly well calibrated initial, non-sequential probabilities. A natural question to ask would be if we can create a calibrated form of sequential learning. This would be done either with a modification of the current algorithm or using additional data to move from a two-step to a three-step procedure that uses more data to calibrate the output. However, in doing this it will be important to understand the relationship between calibration and incorporating the sequential information, as we have seen that with a sequential model the additional information provided does yield true probabilities closer to the edges.

On the application side, we see that in practice sequential methods have both benefits and drawbacks. A particular area for improvement is that they all seem to yield longer blocks of a given state than would be observed in the data — “smoothing” indeed, but not the kind we want. Developing a method that can handle these short bout durations well, whether by selectively choosing when to smooth over parts of the series or using the sequential part of the model in a more sophisticated way would potentially be of use. Another potential solution that would be useful in general is the collection of better data that would yield improved initial estimates. As we saw, sequential methods are highly depending on the probability estimates they start from, and low resolution video using crude features yields a very noisy signal. Improved video data will give sequential methods a better starting point,
or, as has been hinted from the performance on two state data, could eliminate the need for it entirely.

Finally, there is still work to be done on the marginal probability reweighting method. While we have a proof for its benefits under log loss, we are left to ask if similar results can be proved for other loss functions. At present, the best guess is that such results will require additional constraints of special versions of the method and that reweighting may not guarantee universal improvement in all situations. This, too, would be interesting to understand.
BIBLIOGRAPHY


[55] Cosma Rohilla Shalizi and Kristina Lisa Shalizi. Blind construction of optimal nonlin-


