Dynamic Machine Learning with Least Square Objectives

Şan Gültekin

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

Columbia University

2019

ABSTRACT

Dynamic Machine Learning with Least Square Objectives

Şan Gültekin

As of the writing of this thesis, machine learning has become one of the most active research fields. The interest comes from a variety of disciplines which include computer science, statistics, engineering, and medicine. The main idea behind learning from data is that, when an analytical model explaining the observations is hard to find—often in contrast to the models in physics such as Newton's laws—a statistical approach can be taken where one or more candidate models are tuned using data.

Since the early 2000's this challenge has grown in two ways: (i) The amount of collected data has seen a massive growth due to the proliferation of digital media, and (ii) the data has become more complex. One example for the latter is the high dimensional datasets, which can for example correspond to dyadic interactions between two large groups (such as customer and product information a retailer collects), or to high resolution image/video recordings.

Another important issue is the study of dynamic data, which exhibits dependence on time. Virtually all datasets fall into this category as all data collection is performed over time, however I use the term dynamic to hint at a system with an explicit temporal dependence. A traditional example is target tracking from signal processing literature. Here the position of a target is modeled using Newton's laws of motion, which relates it to time via the target's velocity and acceleration.

Dynamic data, as I defined above, poses two important challenges. Firstly, the

learning setup is different from the standard theoretical learning setup, also known as Probably Approximately Correct (PAC) learning. To derive PAC learning bounds one assumes a collection of data points sampled independently and identically from a distribution which generates the data. On the other hand, dynamic systems produce correlated outputs. The learning systems we use should accordingly take this difference into consideration. Secondly, as the system is dynamic, it might be necessary to perform the learning online. In this case the learning has to be done in a single pass. Typical applications include target tracking and electricity usage forecasting.

In this thesis I investigate several important dynamic and online learning problems, where I develop novel tools to address the shortcomings of the previous solutions in the literature. The work is divided into three parts for convenience. The first part is about matrix factorization for time series analysis which is further divided into two chapters. In the first chapter, matrix factorization is used within a Bayesian framework to model time-varying dyadic interactions, with examples in predicting user-movie ratings and stock prices. In the next chapter, a matrix factorization which uses autoregressive models to forecast future values of multivariate time series is proposed, with applications in predicting electricity usage and traffic conditions. Inspired by the machinery we use in the first part, the second part is about nonlinear Kalman filtering, where a hidden state is estimated over time given observations. The nonlinearity of the system generating the observations is the main challenge here, where a divergence minimization approach is used to unify the seemingly unrelated methods in the literature, and propose new ones. This has applications in target tracking and options pricing. The third and last part is about cost sensitive learning, where a novel method for maximizing area under receiver operating characteristics curve is proposed. Our method has theoretical guarantees and favorable sample complexity. The method is tested on a variety of benchmark datasets, and also has applications in online advertising.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First of all I would like to express my gratitude to my advisor Prof. John Paisley. Without his encouragement throughout my studies, this thesis would have not been possible. I am also indebted to Professors David Blei, Daniel Hsu, John Wright, and Xiaodong Wang for being part of my thesis committee and providing invaluable advice.

The long Ph.D. marathon would also have not been possible without my friends—quite a long list which I shall refrain from writing here, for the sole fear of forgetting someone.

Last but by no means the least, I would like to express my gratitude to my family, in particular my parents Akde Gültekin and Bilgin Gültekin, whom this thesis is dedicated to.

To my family

Chapter 1

Introduction

In many areas of science, one is concerned with finding a mathematical model which explains observed phenomena. Perhaps one of the first models to come in mind is Newton's laws of motion. In this case, there is a formula which explains how a physical systems works; for example Newton's second law asserts the acceleration is directly proportional to the applied force. Now consider a setup where one is concerned with movies a person is interested in watching. Can we come up with a formula, similar to $F = ma$ which can tell us if a person will like a given movie? It is unlikely that we can find a closed-form solution to this problem, or even formulate one, as we don't even have fundamental quantities (e.g. mass and acceleration in physics) to relate one's preference to.

The difficulty of finding the true model is not confined to applications in social sciences. In telecommunication systems, for instance, a simple channel with thermal noise can be modeled as Gaussian [51]; however, it is harder to do so for channels with multipath and shadowing. Another example is the Ising model in physics [112] where there are an exponential number of spin configurations. When it is difficult to identify the true model, it is customary to take a *statistical* approach where a candidate model is tuned based on the observations available. In the telecommunication example, one may transmit symbols several times and based on the outputs, the parameters of a

Gaussian mixture distribution can be tuned, which then becomes a model for the channel. This statistical tuning process is also referred to as statistical machine learning, or machine learning, as it is a computationally intensive task and typically done using computers.

The machine learning approach has proved useful in various fields such as computer vision, robotics, medicine, recommender systems, and finance. The main aim is to apply statistical techniques to extract useful information from the available data; which further divides into several categories. Arguably the most common one is where a set of inputs and outputs are given which are used to learn a mapping that can be used to predict future outcomes. This is referred to as supervised learning, as outputs are provided (supervised by the data provider). Another widely occurring one is unsupervised learning, where only the inputs are given and the aim is to discover structure within it. The problem of density estimation can be seen as an example of this. While there are a number of other important subfields such as reinforcement learning, semi-supervised learning, and transfer learning, they are outside the scope of this thesis so I will not further discuss their details here.

Regardless of the kind of data provided, it is safe to say that virtually all datasets have a dependence on time. This is, first of all, because all the collection process happens over time. This might be explicitly available in the dataset as time stamps, or might be discovered during unsupervised learning. The main focus of this thesis is on the former, where an explicit dependence on time is given; we call these dynamic data. Analysis of data, or outputs that are generated dynamically manifests itself in many different fields. In control theory, an important application is the design of controllers that stabilize dynamic plants, in radar target tracking the position of a moving target has to be estimated continually, and in recommender systems a user's changing interests should be taken into account.

In this thesis I develop novel methods which address the learning problems for dynamic datasets and dynamic systems. As the decades have seen a massive growth

in data, one of my main focuses here is scalability. This brings us to the online learning setting. Online learning is a subfield where the learning is applied to a stream of data, in a single pass, without storing the data in its entirety. This is useful for both scalability and real-time application purposes. In addition, it is a natural choice for dynamic datasets. One example is radar target tracking, where the location of the target has to be known in real time. I will refer to online learning and dynamic learning interchangeably.

In Part I, I start with Dynamic Matrix Factorization. I show that matrix factorization is a useful tool for analyzing high dimensional time series. In Chapter 1, I consider dyadic time series, where at every time instant we have a matrix which shows the interaction between two groups. Such matrices are typically high dimensional and sparse, and a low rank matrix factorization approach typically works well. I show how matrix factorization can be integrated with a stochastic process model for the factors, which gives a *generative model* for the data. The resulting model can then be inferred using variational inference, a tool for approximate posterior computation. In Chapter 2, my focus is on forecasting future values of high dimensional and sparse time series, once again using matrix factorization. The main difference here is that, unlike Chapter 1, the observations are a single column vector, so a different generative model is necessary. The analysis here shows important connections between generative models, linear systems, and convexity. In Part II, based on the applications of generative models in Part I, I investigate the more general problem of nonlinear Kalman filtering, which Chapter 1 is an instance of. In Chapter 3, I present nonlinear Kalman filtering with divergence minimization, which uses information theoretic divergence measures to do posterior inference. This results in new filters, and also provides a new perspective on many different existing filters in literature, developed since 1960's. Finally in Part III, I consider a different problem, maximizing the area under receiver operating characteristics curve (AUC). Initially developed for radar detection problems, AUC maximization is widely adopted in cost sensitive learning problems, as it

directly measures the separation ability of a ranking function between two classes. As the metric itself is NP-hard to optimize, I investigate a specific convex relaxation approach which yields a learning-rate free and dynamic learning algorithm. The main difference here is that the data we consider is not necessarily dynamic; however the algorithm can still be applied in a dynamic fashion. Furthermore, using only mild assumptions about the provided data I derive favorable sample complexity bounds for the proposed algorithm.

While the models and applications in this thesis have their own specifics, they can be unified in terms of the cost function they use. In particular, this is the least-squares type cost[1] upon which our algorithms are derived. For Chapter 1, the error made in predictions is penalized as

$$\epsilon_{\text{chap.1}} = \left\| x_{ij}[t] - \boldsymbol{u}_i[t]^\top \boldsymbol{v}_j[t] \right\|_2^2 \tag{1.1}$$

where x_{ij} is the (i,j)-th entry of the observation matrix at time t, and $\boldsymbol{u}_i[t]$ and $\boldsymbol{v}_j[t]$ are the hidden variables generating it. Equivalently we assume that the observation is distributed as Gaussian given the hidden variables. For Chapter 2 the error is

$$\epsilon_{\text{chap.2}} = \left\| \boldsymbol{x}[t] - \boldsymbol{U}[t]^\top \boldsymbol{v}[t] \right\|_2^2 \tag{1.2}$$

where $\boldsymbol{x}[t]$ is the observed vector, $\boldsymbol{U}[t]$ is the reconstruction matrix, and $\boldsymbol{v}[t]$ is the compressed time series vector at time t. Chapter 3 considers the nonlinear Kalman filtering problem, where the observation likelihood is Gaussian with a nonlinear dependency on the hidden variable.

$$\epsilon_{\text{chap.3}} = \left\| \boldsymbol{y}[t] - \boldsymbol{h}(\boldsymbol{x}[t]) \right\|_2^2 \tag{1.3}$$

where $\boldsymbol{y}[t]$ is the observation and $\boldsymbol{x}[t]$ is the hidden variable. Finally in Chapter 4 we

[1] It is "least-squares type" as it is not necessarily least squares. For example when there is an expectation over the unknown variable we will instead have mean square error; then again since the mathematical structure is similar I simply refer to this family of errors as least squares error.

have

$$\epsilon_{\text{chap.4}} = \left[1 - \boldsymbol{w}^\top (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)\right]^2 \tag{1.4}$$

where $\boldsymbol{w}$ is the weight vector we want to learn, and $\boldsymbol{x}$'s are the feature vectors. The specific structure of the least squares error bring important benefits in every chapter: In Chapter 1 it allows for closed form variational inference, in Chapter 2 it yields a convex program with closed form solution, in Chapter 3 it is used to get efficient posterior computation, and in Chapter 4 it gives a learning-rate free, distributed, and asynchronous algorithm.

Part I

Dynamic Matrix Factorization

Chapter 2

Dynamic Matrix Factorization for Dyadic Time Series

2.1 Introduction

Time series are abundant in real life. As I mentioned in the introduction, virtually every data collected is a time series, as the collection happens over time. In this section however, our focus shall be on a specific subset called dyadic time series. For this let us review dyadic datasets first. Such datasets comprise interaction of two groups. Some concrete examples are websites such as Netflix, Amazon, or Youtube where the user is rating movies, items, or videos. In this case the interaction is between the set of users and items. In sporting events the interactions may correspond to the matches between teams (football) or individuals (tennis). In finance, on the other hand, the foreign exchange rates are determined by two countries' economies.

Given the wide availability of dyadic datasets it is important to develop techniques that can leverage their pairwise interaction nature. One successful technique from recommender systems literature is collaborative filtering [100]. Recommender systems aim at finding the interests of the users, given the previous user-item interactions. For example, in case of Netflix a recommender system tries to find movies a user would

be interested in watching. The collaborative filtering approach to this problem is to determine a user's interests based on the available information of all other users. This way, the user histories collaborate to find a meaningful pattern in the data. Although rooted in recommender systems literature, the idea of collaborative filtering is quite general and can be applied to any dyadic dataset.

A particularly interesting collaborative filtering method is that of matrix factorization [70], [98]. Given the key observation that the user-item interactions can be represented by a matrix, we are concerned with finding a user and an item matrix, which factorizes the observation matrix. The individual columns of the factor matrices are then appropriately named user or item factors/embeddings. A key for efficient computation and good prediction performance is the factorization rank. In ideal cases, it can be shown that the sparsely sampled observation matrix can be recovered exactly [20]. In practice, the matrix factorization proved to be useful as well [70].

The dyadic datasets and matrix factorization method we described so far are all static, meaning that there is only a single interaction between any given pair. While this might be true for some cases, there are many datasets for which, this gives a rather unnatural interpretation. For example a static model would assume a person's taste of movies does not change over time, or the outcome of a match between two teams would be the same irrespective of the time it takes place. In practice such models learn a single embedding using all data available, disregarding the time information provided; however from these examples it is evident that a dynamic model could be a better fit. With that said the vast majority of literature is confined to the static setting [70], [98], [99], [79].

A particularly interesting set of models that bring matrix factorization and Bayesian modeling together are probabilistic matrix factorization [98] and Bayesian probabilistic matrix factorization [99]. Here the latent factors are assigned a Gaussian prior, along with a Gaussian likelihood. Inference in the resulting model leads to coordinate

ascent updates for the factors, analogous to the alternating least squares technique used in [70]. The Gaussian interpretation of the Bayesian setting is useful, however, for our purposes.

In this chapter we develop a Bayesian dynamic matrix factorization method, which we call Collaborative Kalman Filter (CKF) where the latent factors are evolving with respect to a multidimensional Brownian motion [29]. This random process model, in addition to the Gaussian likelihood for observations now define a state-space model [113]. Similar to the Kalman filters and motivated by scalable inference, we apply online learning to our model and find all the latent embeddings in a single pass; this makes the algorithm suitable for very large datasets. Since the observations are the inner product of two time-evolving latent variables the likelihood function contains a bilinear term and the resulting state-space model is nonlinear. Unlike the linear-Gaussian case then, here we do not have a closed form solution for the posterior distribution. We therefore leverage variational inference for approximate posterior computation; which is a principled method that maximizes a lower bound on the marginal likelihood of observations [112]. All updates are in closed form, which makes the algorithm structurally similar to the alternating minimization schemes. The overall model is a nonlinear Kalman filter, where the factors are learned collaboratively, hence the name CKF.

The method introduced so far provides a principled way of finding latent embeddings from dyadic time series data, however it does not take parameter estimation into account. For datasets such as stock prices we would like to refine our estimate of Brownian motion drift, to account for the changes in volatility. We also provide a solution to this problem here, where the Brownian motion parameter is estimated using the variational objective function. The resulting filter can then learn the drift from the data, along with embeddings.

The rest of this chapter is organized as follows. In Section 2.2 we overview matrix factorization and Kalman filtering. Section 2.3 introduces the CKF model and Section

2.4 derives variational inference for approximate posterior computation. Section 2.5 lists CKF algorithms and their computational complexity. We show experimental results in Section 2.6 and conclude in Section 2.7.

2.2 Background

The collaborative Kalman filter is based on matrix factorization and Kalman filtering, which we review here for completeness. Based on this we will derive our model in the next section.

2.2.1 Matrix Factorization

Matrix factorization refers to representing a matrix as a product of several factor matrices of certain structure. We shall focus on matrix factorization in the context of collaborative filtering here. Extending the analogy from Section 2.1, let $\boldsymbol{X}$ be an $M \times N$ matrix of observations. Here the rows of $\boldsymbol{X}$ correspond to users and the columns correspond to items. We are interested in representing $\boldsymbol{X}$ as a product of a $d \times M$ matrix $\boldsymbol{U}$ and a $d \times N$ matrix $\boldsymbol{V}$, i.e. $\boldsymbol{X} \approx \boldsymbol{U}^\top \boldsymbol{V}$. It is then clear that the columns of $\boldsymbol{U}$ (i.e. $\boldsymbol{u}_i$) are representations of users in $\mathbb{R}^d$, and the columns of $\boldsymbol{V}$ (i.e. $\boldsymbol{v}_j$) are representations of items, in the same $\mathbb{R}^d$. Any given observation is approximated by the inner product $x_{ij} \approx \boldsymbol{u}_i^\top \boldsymbol{v}_j$

While M and N are determined by the observation matrix $\boldsymbol{X}$, d is a free parameter introduced by factorization. While there is not a single choice for this parameter, $d \ll \min\{M, N\}$ is desirable for two reasons: (i) it provides regularization which prevents overfitting, and (ii) it makes the learning algorithm computationally efficient. Typically the observation matrix has very large dimensions, for example a retailer can have millions of customers and tens of thousands of items for sale, but a given customer only interacts with a very small fraction of items. Then the matrix will have a full rank partitioning [41] where there are at most $d \ll \min\{M, N\}$ linearly

independent rows and columns. Note my choice of notation here; in the ideal case we would like to choose the free parameter d such that it matches the true rank of $\boldsymbol{X}$.

The low rank factorization, when applied to collaborative filtering, can be regarded as classical (or frequentist) statistical inference. To see this suppose that the observations are coming from a probability distribution: $x_{ij} \sim p(\boldsymbol{u}_i^T \boldsymbol{v}_j)$. For example, when x is binary, $p(\cdot)$ could be the logistic or probit likelihood, when m-ary it could be the ordered probit, and when real-valued it could be a univariate Gaussian. Note that here the factors are treated as unknown constants, as is the case with classical inference. We next touch upon the connection between this model and alternating least squares. Suppose that $x_{ij} \sim N(\boldsymbol{u}_i^T \boldsymbol{v}_j, \sigma^2)$, where we imposed univariate Gaussian as the likelihood function. Using maximum likelihood inference approach the update of $\boldsymbol{u}_i$ is obtained using the relevant $\boldsymbol{v}_j$ and x_{ij}. If we let the set $\Omega_{\boldsymbol{u}_i}$ to contain the index values of objects that user i has rated, the update is

$$\begin{aligned} \boldsymbol{u}_i &= \left(\sum_{j \in \Omega_{\boldsymbol{u}_i}} \boldsymbol{v}_j \boldsymbol{v}_j^T\right)^{-1} \left(\sum_{j \in \Omega_{\boldsymbol{u}_i}} x_{ij} \boldsymbol{v}_j\right) \\ \boldsymbol{v}_j &= \left(\sum_{i \in \Omega_{\boldsymbol{v}_j}} \boldsymbol{u}_i \boldsymbol{u}_i^T\right)^{-1} \left(\sum_{i \in \Omega_{\boldsymbol{v}_j}} x_{ij} \boldsymbol{u}_i\right) \end{aligned} \tag{2.1}$$

and the update for $\boldsymbol{v}_j$ is symmetric.

We see that Eq (2.1) is identical to the alternating least square updates [70]. In the case of probabilistic matrix factorization [98], the latent factors are assigned a Gaussian prior. Using spherical priors $\forall i : \boldsymbol{u}_i \sim N(0, \lambda_u^{-1} \boldsymbol{I})$ and $\forall j : \boldsymbol{v}_j \sim N(0, \lambda_v^{-1} \boldsymbol{I})$ the updates are

$$\begin{aligned} \boldsymbol{u}_i &= \left(\lambda_u \boldsymbol{I} + \sum_{j \in \Omega_{\boldsymbol{u}_i}} \boldsymbol{v}_j \boldsymbol{v}_j^T\right)^{-1} \left(\sum_{j \in \Omega_{\boldsymbol{u}_i}} x_{ij} \boldsymbol{v}_j\right) \\ \boldsymbol{v}_j &= \left(\lambda_v \boldsymbol{I} + \sum_{i \in \Omega_{\boldsymbol{v}_j}} \boldsymbol{u}_i \boldsymbol{u}_i^T\right)^{-1} \left(\sum_{i \in \Omega_{\boldsymbol{v}_j}} x_{ij} \boldsymbol{u}_i\right) \end{aligned} \tag{2.2}$$

The updates are once again in closed form and structurally very similar to alternating least squares. Introducing the parameters λ_u and λ_v are useful, as they regularize the solutions, furthermore they are necessary when the outer product matrices are poorly conditioned. In fact, this term is typically added when computing least squares solutions, without explicit mention to the Bayesian interpretation.

2.2.2 Kalman filtering

Given the matrix factorization approach to collaborative filtering, the question is how to make it a dynamic model. Given that the users and items are represented as vectors in $\mathbb{R}^d$ it is natural to view them as realizations of a stochastic process. In particular we will be using multidimensional Brownian motion to characterize the temporal dynamics of latent factors, which along with the likelihood function of Eq. (2.2) constitutes a state-space model [48]. This is in contrast to solely using Eq. (2.2), which would disregard time information. Given the state-space model we naturally seek a solution using variations of Kalman filter, which is optimal for the linear-Gaussian state-space models [113]. (In fact, Kalman's original formulation [66] is optimal for a more general class of noise distributions; but from the Bayesian viewpoint [97] the emphasis is on linear-Gaussian model since the Kalman filter computes the exact posterior in this case.) In this section we review the Kalman filter based on a parametrization that will be useful for the subsequent development of our CKF.

The Kalman filter models a sequence of observed vectors $\boldsymbol{y}_t \in \mathbb{R}^p$ as linear functions of a sequence of latent state vectors $\boldsymbol{x}_t \in \mathbb{R}^d$ with additive noise. These state vectors evolve according to a first-order Markov process, where the current state equals the previous state plus additive noise. Assuming a Gaussian prior distribution on $\boldsymbol{x}_0$, then for $t = 1, \ldots, T$ and zero-mean noise, this is written as follows,

$$\begin{aligned} \boldsymbol{x}_t \,|\, \boldsymbol{x}_{t-1} &\sim N(\boldsymbol{x}_{t-1}, \alpha \boldsymbol{I}) \\ \boldsymbol{y}_t \,|\, \boldsymbol{x}_t &\sim N(\boldsymbol{A}\boldsymbol{x}_t, \sigma^2 \boldsymbol{I}). \end{aligned} \tag{2.3}$$

Inference for the state dynamics of $\boldsymbol{x}$ proceeds according to the forward algorithm [1], which includes two steps: (i) marginalizing the previous state to obtain a prior distribution on the current state, and (ii) calculating the posterior of the current state given an observation.

[1] There is also the forward-backward algorithm, which is used for Kalman smoothing. However, since the focus of this chapter is online learning, we do not discuss this extension any further.

Specifically, let the distribution of the previous state be $\boldsymbol{x}_{t-1} \sim N(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$. The distribution of the current state $\boldsymbol{x}_t$ is calculated by marginalizing the previous state,

$$\begin{aligned} p(\boldsymbol{x}_t) &= \int_{\mathbb{R}^d} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) p(\boldsymbol{x}_{t-1}) d\boldsymbol{x}_{t-1} \\ &= N(\boldsymbol{x}_t|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1} + \alpha \boldsymbol{I}). \end{aligned} \tag{2.4}$$

The free parameter α is a measure of drift in one unit of time and can be interpreted as the volatility of the state vectors. We will later present a method for dynamically estimating this parameter on-the-fly.

After observing $\boldsymbol{y}_t$, the posterior of $\boldsymbol{x}_t$ is

$$\begin{aligned} p(\boldsymbol{x}_t|\boldsymbol{y}_t) &\propto p(\boldsymbol{y}_t|\boldsymbol{x}_t) p(\boldsymbol{x}_t) \\ &= N(\boldsymbol{x}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \end{aligned} \tag{2.5}$$

where, after defining $\boldsymbol{B}_t = \boldsymbol{\Sigma}_{t-1} + \alpha I$,

$$\begin{aligned} \boldsymbol{\Sigma}_t &= \left[\boldsymbol{A}^\top \boldsymbol{A}/\sigma^2 + \boldsymbol{B}_t^{-1}\right]^{-1} \\ \boldsymbol{\mu}_t &= \boldsymbol{\Sigma}_t \ \left[\boldsymbol{B}_t^{-1}\boldsymbol{\mu}_{t-1} + \boldsymbol{A}^\top \boldsymbol{y}_t/\sigma^2\right]. \end{aligned} \tag{2.6}$$

These posterior updates immediately follow from Bayes' rule. An alternative viewpoint is to find the linear minimum mean square error estimator (LMMSE) for $\boldsymbol{x}_t$. Both formulations yield the same result, as for the Gaussian distribution the mean of the posterior and LMMSE coincide. With that said, the formulae in Eq. (2.6) look different from the Kalman gain-based state-space updates [113]. However the two are in fact equivalent, and the classical formulae can be obtained from Eq. (2.6) using matrix inversion lemma [41]. We will go into details of this when we discuss the nonlinear Kalman filtering problem in Part II.

Following Eq. (2.4), we can use this posterior distribution of $\boldsymbol{x}_t$ in Eq. (2.6) to find the prior for the next state $\boldsymbol{x}_{t+1}$. Since the initial distribution is Gaussian, for each value of t, the prior and posterior has Gaussian distribution that are given by Eqs. (2.4) and (2.6) respectively.

Our derivation above is for discrete and homogeneous time indices, i.e. $t \in \{0, 1, \ldots, T\}$. The extension to continuous time only requires a simple modification: For continuous-time Kalman filters, the variance of the drift from $\boldsymbol{x}_{t-1}$ to $\boldsymbol{x}_t$ depends on the time between these two events. Calling this time difference Δ_t, we can make the continuous-time extension by replacing αI with $\Delta_t \alpha I$, in which case $\boldsymbol{x}_t$ becomes a Brownian motion [29].

2.3 Collaborative Kalman Filter (CKF)

In this section we develop the generative model for the dynamic dyadic time series, which constitute the CKF. The motivation is to extend the matrix factorization model in described in Section 2.2.1 to the dynamic setting, using the continuous-time Kalman filtering framework discussed in Section 2.2.2. The state-space model provides a natural means to do so; we model each latent location using a Brownian motion in $\mathbb{R}^d$.

The set of latent factors are denoted by $\{\boldsymbol{u}_i[t]\}_{i=1}^M$ and $\{\boldsymbol{v}_j[t]\}_{j=1}^N$, where the time subscript is now added to show time dependence. For instance the variables $\boldsymbol{u}_i[t]$ and $\boldsymbol{v}_j[t]$ represent the user i and item j at time t, in that order. We use the likelihood model of Section 2.2.1 to relate the observation to the factors, in particular, a given observation $x_{ij}[t]$ is now a random variable parametrized by the inner product $\langle \boldsymbol{u}_i[t], \boldsymbol{v}_j[t] \rangle$. The exact form of the likelihood depends on the observation type, for continuous observations such as stock prices the likelihood is a simple Gaussian, whereas for ordinal observations such as movie ratings, an ordered probit likelihood model [44] is used. For the derivations we shall focus on a single observation (rating, price, etc.) for brevity; which will then be generalized to multiple observations. The extension is quite straightforward.

Likelihood model (continuous): For continuous observations $x_{ij}[t] \in R$ is modeled as a draw from a Gaussian distribution with the mean $\langle \boldsymbol{u}_i[t], \boldsymbol{v}_j[t] \rangle$ and some preset

variance σ^2. The conditional probability is written as

$$P(x_{ij}[t] \mid \boldsymbol{u}_i, \boldsymbol{v}_j) = N\left(x_{ij}[t] \mid \langle \boldsymbol{u}_i[t], \boldsymbol{v}_j[t] \rangle, \sigma^2\right). \tag{2.7}$$

The likelihood in Eq. (2.7), along with spherical Gaussian priors on $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ is identical to the joint likelihood of probabilistic matrix factorization when the time indices are dropped.

Likelihood model (ordinal): For m-class ordinal ordinal observations, the observation $x_{ij}[t] \in \{1, \ldots, m\}$, obtained by first drawing from a Gaussian distribution with the mean $\langle \boldsymbol{u}_i[t], \boldsymbol{v}_j[t] \rangle$ and variance σ^2, and then funneling the outcome through a continuous-to-discrete map. To obtain the map, the real line $\mathbb{R}$ is partitioned into m regions with each region denoting a class such as, for example, star ratings for movies. Let $\mathcal{I}_k = (l_k, r_k]$ be the partition for class k where $l_k < r_k$, $r_k = l_{k+1}$, $l_k = r_{k-1}$ and $k = 1, \ldots, m$. Therefore, the model assumes an order relation between the m classes, for example that a 4-star rating is closer to a 5-star rating than a 1-star rating. (In the binary special case, this order relation no longer matters.) Then, the probability that $x_{ij}[t] = k$ is

$$P(x_{ij}[t] = k \mid \boldsymbol{u}_i[t], \boldsymbol{v}_j[t]) = \int_{\mathcal{I}_k} N\left(y_{ij}[t] \mid \langle \boldsymbol{u}_i[t], \boldsymbol{v}_j[t] \rangle, \sigma^2\right) dy. \tag{2.8}$$

Unlike the continuous model, the inference will not be in closed form if the observation $x_{ij}[t]$ depends directly on the factors. This is solved by introducing auxiliary variables, similar to the Expectation Maximization algorithm [13]. The auxiliary variable $y_{ij}[t]$ is a continuous variable parametrized by the mean $\langle \boldsymbol{u}_i[t], \boldsymbol{v}_j[t] \rangle$, which in a sense equivalent to the continuous observation we described before. Given this latent variable, the actual observation is obtained through the continuous to discrete map. This yields the following hierarchy

$$\begin{aligned} x_{ij}[t] \mid y_{ij}[t] &= \textstyle\sum_k k \; \mathbb{I}(y_{ij}[t] \in \mathcal{I}_k), \\ y_{ij}[t] \mid \boldsymbol{u}_i[t], \boldsymbol{v}_j[t] &\sim N(\langle \boldsymbol{u}_i[t], \boldsymbol{v}_j[t] \rangle, \sigma^2). \end{aligned} \tag{2.9}$$

It can be seen that, given the latent variable the observation is deterministic, so all randomness is absorbed into $y_{ij}[t]$. Compared to the static case, dynamic modeling allows a pair to interact more than once, as for every value of t, there can be a different $x_{ij}[t]$. In the static case, each pair would only be allowed a fixed outcome, however as we mentioned this is rather unnatural for many types of data. Therefore, when we have dyadic time series data, CKF can model it effectively.

Prior model: We now describe the priors for the latent factors $\{\boldsymbol{u}_i[t]\}_{i=1}^{M}$ and $\{\boldsymbol{v}_j[t]\}_{j=1}^{N}$. For the PMF model these were spherical Gaussians with fixed variance, which makes sense for a static model. For the dynamic case we will be using multi-dimensional Brownian motion. Let the duration of time since the last event be $\Delta_{\boldsymbol{u}_i}^{[t]}$ for $\boldsymbol{u}_i$ and $\Delta_{\boldsymbol{v}_j}^{[t]}$ for $\boldsymbol{v}_j$. Also, as with the Kalman filter, assume that the posterior distributions at previous time are multivariate Gaussian: In other words, we consider $\boldsymbol{u}_i[t-\Delta_{\boldsymbol{u}_i}^{[t]}]$ and $\boldsymbol{v}_j[t-\Delta_{\boldsymbol{v}_j}^{[t]}]$, with $t-\Delta_{\boldsymbol{u}_i}^{[t]}$ and $t-\Delta_{\boldsymbol{v}_j}^{[t]}$ being the time of the last observation for $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ respectively. Then their distribution can be written as

$$\begin{aligned}\boldsymbol{u}_i[t-\Delta_{\boldsymbol{u}_i}^{[t]}] &\sim N(\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}]\ ,\ \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}])\\ \boldsymbol{v}_j[t-\Delta_{\boldsymbol{v}_j}^{[t]}] &\sim N(\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t-\Delta_{\boldsymbol{v}_j}^{[t]}]\ ,\ \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t-\Delta_{\boldsymbol{v}_j}^{[t]}]).\end{aligned} \tag{2.10}$$

Since the latent factors are time varying, their mean and covariance parameters are also indexed by time. At any given time t, a factor will have two sets of parameters, the prior $(\boldsymbol{\mu}_\bullet[t], \boldsymbol{\Sigma}_\bullet[t])$ and the posterior $(\boldsymbol{\mu}'_\bullet[t], \boldsymbol{\Sigma}'_\bullet[t])$, where we use the prime sign to distinguish the posterior, and $\bullet$ represents an arbitrary factor. We can now apply Eq. (2.4) to marginalize $\boldsymbol{u}_i[t-\Delta_{\boldsymbol{u}_i}^{[t]}]$ and $\boldsymbol{v}_j[t-\Delta_{\boldsymbol{v}_j}^{[t]}]$ in their respective interval $[t-\Delta_\bullet^{[t]}\ ,\ t]$ and obtain the prior distributions of $\boldsymbol{u}_i[t]$ and $\boldsymbol{v}_j[t]$

$$\begin{aligned}\boldsymbol{u}_i[t] &\sim N(\boldsymbol{\mu}_{\boldsymbol{u}_i}[t]\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t])\\ \boldsymbol{v}_j[t] &\sim N(\boldsymbol{\mu}_{\boldsymbol{v}_j}[t]\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t])\end{aligned} \tag{2.11}$$

where

$$\boldsymbol{\mu}_\bullet[t] = \boldsymbol{\mu}'_\bullet[t-\Delta_\bullet^{[t]}] \quad , \quad \boldsymbol{\Sigma}_\bullet[t] = \boldsymbol{\Sigma}'_\bullet[t-\Delta_\bullet^{[t]}] + \Delta_\bullet^{[t]}\alpha\boldsymbol{I} \tag{2.12}$$

for the respective $\boldsymbol{u}_i[t]$ or $\boldsymbol{v}_j[t]$.

With this formulation, CKF is a Kalman filter with multiple state vectors that interact at the observation, therefore it is different from the traditional setup of Eq. (2.3) where the entire state can be described by a single variable. We also recall from Section 2.2.2 that $\alpha \boldsymbol{I}$ is the drift covariance in one unit of time, and so the transition from posterior to prior involves the addition of a small value to the diagonal of the posterior covariance matrix. The positive value of this parameter is what allows for dynamic modeling, and when $\alpha = 0$, CKF reduces to online inference for a static model, as the latent factors become fixed.

Hyperprior model: The drift parameter α controls how much the state vectors can move in one unit of time. When α becomes bigger, the state vectors can move greater distances to better fit the observation x; this allows better modeling for fast changing observations but it also makes the learned vectors more forgetful of previous information. On the other extreme, as $\alpha \to 0$ the model does not forget any previous information and the state vectors simply converge to a point, as in this case we are simply doing online inference for a static model.

We develop the CKF model by allowing α to dynamically change in time as well, which we can learn on-the-fly. This leads to interesting analyses, particularly for stock price data, where the model can learn volatility in an unsupervised manner. We present this in Section 2.6. For notational convenience, we define the hyperprior for a shared α, but the extensions to a $\boldsymbol{u}_i$ or $\boldsymbol{v}_j$ specific α is straightforward, which we will discuss in more detail in Section 2.4.

We model α as a geometric Brownian motion by defining $\alpha[t] = e^{a[t]}$ and letting $a[t]$ be a Brownian motion. Therefore, as with the state vectors, if $t - \Delta_a^{[t]}$ is the last observed time for $a[t]$, the distribution of $a[t]$ is

$$a[t] \sim N(a[t - \Delta_a^{[t]}] \ , \ c \ \Delta_a^{[t]}). \tag{2.13}$$

Again there is a drift parameter c that plays an important role and requires a good setting, but we observe that defining α to be an exponentiated Brownian motion has

an important modeling purpose by allowing for time-varying volatility. In Section 2.4.2 we show how c can be inferred conveniently from the variational approximation.

Finally, since $\alpha[t]$ is modified to be a geometric Brownian motion, the transition from previous posterior to current prior for the states needs to be adjusted accordingly. In Equation (2.11) the constant value of α allowed for $\boldsymbol{\Sigma}_\bullet[t]$ to be calculated in closed form, particularly by adding a time-scaled $\alpha \boldsymbol{I}$ to the previous posterior. In this case, the update is modified for any given factor by performing the integration implied in Equation (2.11),

$$\boldsymbol{\Sigma}_\bullet[t] = \boldsymbol{\Sigma}'_\bullet[t - \Delta_\bullet^{[t]}] + \boldsymbol{I} \int_{t-\Delta_\bullet^{[t]}}^{t} e^{a[s]} ds. \tag{2.14}$$

We will also derive a simple approximation to this stochastic integral in Section 2.4. Overall for a single pair the generative model of CKF is

$$\begin{aligned} p(y_{ij}[t], \boldsymbol{u}_i[t], \boldsymbol{v}_j[t]) &= p(y_{ij}[t] | \boldsymbol{u}_i[t], \boldsymbol{v}_j[t])\ p(\boldsymbol{u}_i[t])\ p(\boldsymbol{v}_j[t]) \\ &= N(y_{ij}[t] \mid \boldsymbol{u}_i[t]^\top \boldsymbol{v}_j[t], \sigma^2) \times \\ &\quad N(\boldsymbol{u}_i[t] \mid \boldsymbol{\mu}_{\boldsymbol{u}_i}[t], \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t])\ N(\boldsymbol{v}_j[t] \mid \boldsymbol{\mu}_{\boldsymbol{v}_j}[t], \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t]) \end{aligned} \tag{2.15}$$

for continuous observations and

$$\begin{aligned} p(x_{ij}[t], y_{ij}[t], \boldsymbol{u}_i[t], \boldsymbol{v}_j[t]) &= p(z_{ij}[t] \mid y_{ij}[t])\ p(y_{ij}[t] | \boldsymbol{u}_i[t], \boldsymbol{v}_j[t])\ p(\boldsymbol{u}_i[t])\ p(\boldsymbol{v}_j[t]) \\ &= \mathbb{1}(y_{ij}[t] \in \mathcal{I}_{x_{ij}[t]})\ N(y_{ij}[t] \mid \boldsymbol{u}_i[t]^\top \boldsymbol{v}_j[t], \sigma^2) \times \\ &\quad N(\boldsymbol{u}_i[t] \mid \boldsymbol{\mu}_{\boldsymbol{u}_i}[t], \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t])\ N(\boldsymbol{v}_j[t] \mid \boldsymbol{\mu}_{\boldsymbol{v}_j}[t], \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t]) \end{aligned} \tag{2.16}$$

for ordinal observations.

2.4 Variational Inference

Unlike the Kalman Filter of Eq. (2.3) the posterior of CKF is not in closed form, and is difficult to compute. For this reason approximation schemes are required. For Bayesian inference, the standard technique is the Markov Chain Monte Carlo [96], which simulates a Markov Chain that converges to the desired posterior. However,

scalability is an issue with this method, since we usually want to run our algorithms on massive datasets. To this end, compromising the speed advantages of the forward Kalman filter with sampling methods is undesirable.

For this reason we use variational inference, which finds an approximating posterior distribution using optimization. Before we delve into the variational inference for CKF, I first briefly review this technique. In a similar spirit to the previous section let $\boldsymbol{X}$ denote the observations and $\boldsymbol{Z}$ denote the latent variables. For CKF, $\boldsymbol{X}$ would be the observation matrix, and $\boldsymbol{Z}$ would comprise the relevant $\boldsymbol{u}$, $\boldsymbol{v}$, and y. The task is to compute the posterior $p(\boldsymbol{Z}|\boldsymbol{X})$ which for many models of interest is intractable. For example, the likelihood terms of CKF contain a bilinear term, which cannot be integrated in closed form. The aim is then to approximate the true posterior with another distribution which we call $q(\boldsymbol{Z})$. One way of measuring the approximating quality is the forward Kullback-Leibler (KL) divergence [2]

$$KL[q(\boldsymbol{Z})||p(\boldsymbol{Z}|\boldsymbol{X})] = -\int q(\boldsymbol{Z}) \log \frac{p(\boldsymbol{Z}|\boldsymbol{X})}{q(\boldsymbol{Z})} d\boldsymbol{Z} \ . \tag{2.17}$$

However, minimizing this metric directly is not possible as it requires knowledge of $p(\boldsymbol{Z}|\boldsymbol{X})$. To resolve this, the following identity is used

$$\log p(\boldsymbol{X}) = \mathcal{L}[q(\boldsymbol{Z})] + KL[q(\boldsymbol{Z})||p(\boldsymbol{Z}|\boldsymbol{X})] \tag{2.18}$$

called the marginal likelihood of observations. Note that this quantity is a constant, as the data is fixed, and since $KL[q(\boldsymbol{Z})||p(\boldsymbol{Z}|\boldsymbol{X})] > 0$, $\mathcal{L}[q]$ is a lower bound on the marginal likelihood under any q-distribution; this is called the variational lower bound (VLB). Then, as the marginal likelihood is constant, maximizing the lower bound is equivalent to minimizing the KL divergence. A simple algebraic manipulation reveals

[2] Which is called "forward", as swapping the terms give a different metric, called the "reverse". We discuss this difference in more detail when we cover nonlinear Kalman filtering in Part II.

that the lower bound has the following form

$$\begin{aligned}\mathcal{L}[q(\boldsymbol{Z})] &= \int q(\boldsymbol{Z}) \log \frac{p(\boldsymbol{X}, \boldsymbol{Z})}{q(\boldsymbol{Z})} d\boldsymbol{Z} \\ &= \mathbb{E}_q[\log p(\boldsymbol{X}, \boldsymbol{Z})] - \mathbb{E}_q[\log q(\boldsymbol{Z})] \end{aligned} \tag{2.19}$$

where the first term is the expectation of the joint likelihood, and the second term is the entropy of the q-distribution. For many models of interest –including CKF– both terms can be obtained analytically, which resolves the intractability issue (note: $\mathbb{E}_q[\cdot]$ is expectation taken w.r.t q). While $\mathcal{L}[q]$ can be calculated in closed form, the optimal solution would still be $q(\boldsymbol{Z}) = p(\boldsymbol{Z}|\boldsymbol{X})$, which is once again unavailable. So we need to make a simplifying assumption for the posterior. A widely used one is the mean-field approximation [13], where the q-distribution factorizes per component as

$$q(\boldsymbol{Z}) = \prod_i q(\boldsymbol{Z}_i). \tag{2.20}$$

Under mean-field assumption can further write

$$\begin{aligned}\mathcal{L}[q(\boldsymbol{Z})] &= \int q(\boldsymbol{Z}) \left\{\log p(\boldsymbol{X}, \boldsymbol{Z}) - \log q(\boldsymbol{Z})\right\} d\boldsymbol{Z} \\ &= \int \prod_i q(\boldsymbol{Z}_i) \left\{\log p(\boldsymbol{X}, \boldsymbol{Z}) - \sum_i \log q(\boldsymbol{Z}_i)\right\} d\boldsymbol{Z} \end{aligned} \tag{2.21}$$

where for a fixed $q(\boldsymbol{Z}_j)$ we get

$$\mathcal{L}[q(\boldsymbol{Z})] = \int q(\boldsymbol{Z}_j) \log \tilde{p}(\boldsymbol{X}, \boldsymbol{Z}_j) d\boldsymbol{Z}_j - \int q(\boldsymbol{Z}_j) \log q(\boldsymbol{Z}_j) d\boldsymbol{Z}_j. \tag{2.22}$$

We have also defined the probability distribution

$$\log \tilde{p}(\boldsymbol{X}, \boldsymbol{Z}_j) = \mathbb{E}_{i \neq j}[\log p(\boldsymbol{X}, \boldsymbol{Z})] \ + \ \text{const.} \tag{2.23}$$

where the constant term is included for normalization. If we maximize the VLB in

Eq. (2.22) with respect to $q(\boldsymbol{Z}_j)$, keeping the remaining q-distributions fixed, we get

$$\begin{aligned} q^\star(\boldsymbol{Z}_j) &= \arg\min_{q(\boldsymbol{Z}_j)} \int q(\boldsymbol{Z}_j) \log \tilde{p}(\boldsymbol{X}, \boldsymbol{Z}_j) d\boldsymbol{Z}_j - \int q(\boldsymbol{Z}_j) \log q(\boldsymbol{Z}_j) d\boldsymbol{Z}_j \\ &= \arg\min_{q(\boldsymbol{Z}_j)} \int q(\boldsymbol{Z}_j) \log \frac{\tilde{p}(\boldsymbol{X}, \boldsymbol{Z}_j)}{q(\boldsymbol{Z}_j)} d\boldsymbol{Z}_j \\ &= \arg\min_{q(\boldsymbol{Z}_j)} \mathrm{KL}\ [\ \tilde{p}(\boldsymbol{X}, \boldsymbol{Z}_j)\ ||\ q(\boldsymbol{Z}_j)\ ] \\ &= \tilde{p}(\boldsymbol{X}, \boldsymbol{Z}_j) \end{aligned} \tag{2.24}$$

which can be conveniently obtained via

$$q^\star(\boldsymbol{Z}_j) = \frac{\exp\{\mathbb{E}_{i \neq j} \log p(\boldsymbol{X}, \boldsymbol{Z})\}}{\int \exp\{\mathbb{E}_{i \neq j} \log p(\boldsymbol{X}, \boldsymbol{Z})\}}\ . \tag{2.25}$$

In practice, conjugate exponential family models are typically used, where the prior and the posterior of a variable comes from the same exponential family distribution. In this case we can easily obtain the posterior distribution's parameters by taking the expectation $\mathbb{E}_{i \neq j} \log p(\boldsymbol{X}, \boldsymbol{Z})$ and matching the values. This is in fact how we derive the variational inference for CKF, which we tackle next. We break down the derivation into two parts: In the first part we find approximate posteriors for the latent variables $y_{ij}[t]$, $\boldsymbol{u}_i[t]$, and $\boldsymbol{v}_j[t]$ given the observation $x_{ij}[t]$, using variational inference. In the second part we utilize the obtained VLB to infer the drift of the Brownian motion $a[t]$ using Type II maximum likelihood.

2.4.1 Variational Inference for CKF

Based on the key formula of Eq. (2.25) we now derive variational inference for CKF from first principles, focusing on a single observation case. We shall focus on the ordered probit model, as the continuous observation model is just a special case of it. In Section 2.5 we provide full algorithms with update equations for all different cases. For the ordered probit model we have an ordinal observation $x_{ij}[t] \in \{1, \ldots, m\}$, a latent variable $y_{ij}[t]$, and the latent factors $\boldsymbol{u}_i[t]$ and $\boldsymbol{v}_j[t]$. In what follows we drop all time indices to keep the formulae uncluttered, so unless otherwise noted, the time

index is t. The priors on $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ are Gaussian with mean-covariance $(\boldsymbol{\mu}_{\boldsymbol{u}_i}, \boldsymbol{\Sigma}_{\boldsymbol{u}_i})$, and $(\boldsymbol{\mu}_{\boldsymbol{v}_j}, \boldsymbol{\Sigma}_{\boldsymbol{v}_j})$. The exact dependence of current prior on previous posterior will be clear when we discuss the Brownian motion approximation and parameter estimation in Section 2.4.2.

The posterior of interest is $p(y_{ij}, \boldsymbol{u}_i, \boldsymbol{v}_j | x_{ij})$ which is proportional to the joint likelihood, which in turn factorizes as

$$\begin{aligned} p(y_{ij}, \boldsymbol{u}_i, \boldsymbol{v}_j | x_{ij}) &\propto p(x_{ij}, y_{ij}, \boldsymbol{u}_i, \boldsymbol{v}_j) \\ &\propto p(x_{ij}|y_{ij})\ p(y_{ij}|\boldsymbol{u}_i, \boldsymbol{v}_j)\ p(\boldsymbol{u}_i)\ p(\boldsymbol{v}_j)\ . \end{aligned} \tag{2.26}$$

For the posterior distribution we use a mean-field approximation

$$q(y_{ij}, \boldsymbol{u}_i, \boldsymbol{v}_j) = q(y_{ij})\ q(\boldsymbol{u}_i)\ q(\boldsymbol{v}_j) \tag{2.27}$$

The mean-field variational inference is based on treating these three latent variables as independent in the posterior, which gives tractable posteriors. The updates for the variables of course still depend on each other, as they are coupled in the joint likelihood. The corresponding VLB is obtained by

$$\mathcal{L}[q(y_{ij}, \boldsymbol{u}_i, \boldsymbol{v}_j)] = \mathbb{E}_q[\log p(x_{ij}, y_{ij}, \boldsymbol{u}_i, \boldsymbol{v}_j)] - \mathbb{E}_q[\log q(y_{ij}, \boldsymbol{u}_i, \boldsymbol{v}_j)] \tag{2.28}$$

The variational update for y_{ij} is obtained by

$$\begin{aligned} q^\star(y_{ij}) &\propto \mathbb{E}_{i \neq j}\left[\log p(x_{ij}|y_{ij}) + \log p(y_{ij}|\boldsymbol{u}_i, \boldsymbol{v}_j)\right] \\ &\propto \mathbb{E}_{q_{-i}}\left[\log p(x_{ij}|y_{ij}) + \log p(y_{ij}|\boldsymbol{u}_i, \boldsymbol{v}_j)\right] \\ &\propto \mathbb{E}_{q_{-i}}\left[\mathbb{1}(y_{ij} \in \mathcal{I}_{x_{ij}[t]})\right] - \mathbb{E}_{q_{-i}}\left[\frac{1}{2\sigma^2}(y_{ij} - \boldsymbol{u}_i^\top \boldsymbol{v}_j)^2\right] \\ &\propto \mathbb{1}(y_{ij} \in \mathcal{I}_{x_{ij}[t]}) - \frac{1}{2\sigma^2}(y_{ij} - \boldsymbol{\mu}_{\boldsymbol{u}_i}'^\top \boldsymbol{\mu}_{\boldsymbol{v}_j}')^2\ . \end{aligned} \tag{2.29}$$

Several things to note here: Since we only consider proportionality, we can ignore the constant terms as they do not have any effect. The expectations are with respect to every variable in the posterior distribution except y_{ij}. Then the first summand is

not affected by this operation. For the second summand the expectation acts in the inner product $\boldsymbol{u}_i^\top \boldsymbol{v}_j$. Since the expectation is with respect to the approximating posterior we simply get $\boldsymbol{\mu}_{\boldsymbol{u}_i}'^\top \boldsymbol{\mu}_{\boldsymbol{v}_j}'$ as the two factors are independent in the q-distribution. From this functional form we immediately observe the posterior of y_{ij} is a truncated normal

$$q^\star(y_{ij}) = \mathcal{TN}(\boldsymbol{\mu}_{\boldsymbol{u}_i}'^\top \boldsymbol{\mu}_{\boldsymbol{v}_j}',\ \sigma^2,\ \mathcal{I}_{x_{ij}}). \tag{2.30}$$

where the last term gives the boundaries. In this model the observations affect the updates by adjusting these boundaries. In order to update $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ we will need the expectation of y_{ij}. Defining the mean parameter to be

$$m_{ij}' = \boldsymbol{\mu}_{\boldsymbol{u}_i}'^\top \boldsymbol{\mu}_{\boldsymbol{v}_j}' \tag{2.31}$$

the expectation of y_{ij} depends on the interval in which it falls to, which in turn is given by x_{ij}. For $\mathcal{I}_{x_{ij}}$ let $l_{x_{ij}}$ and $r_{x_{ij}}$ be the left and right boundaries of this interval. Then defining

$$\alpha_{ij} = \frac{l_{x_{ij}} - m_{ij}'}{\sigma}, \quad \beta_{ij} = \frac{r_{x_{ij}} - m_{ij}'}{\sigma}$$

we have that

$$\mathbb{E}_q[y_{ij}] = m_{ij}' + \sigma \frac{\phi(\alpha_{ij}) - \phi(\beta_{ij})}{\Phi(\beta_{ij}) - \Phi(\alpha_{ij})}, \tag{2.32}$$

where $\phi(\cdot)$ is the PDF and $\Phi(\cdot)$ the CDF of a standard normal.

The next derivation is for the posterior of $\boldsymbol{u}_i$. We have

$$\begin{aligned}
q^\star(\boldsymbol{u}_i) &\propto \mathbb{E}_{q_{-i}}\left[\log p(y_{ij}|\boldsymbol{u}_i, \boldsymbol{v}_j) + \log p(\boldsymbol{u}_i)\right] \\
&\propto \mathbb{E}_{q_{-i}}\left[-(y_{ij} - \boldsymbol{u}_i^\top \boldsymbol{v}_j)^2 - (\boldsymbol{u}_i - \boldsymbol{\mu}_{\boldsymbol{u}_i})^\top \boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}(\boldsymbol{u}_i - \boldsymbol{\mu}_{\boldsymbol{u}_i})\right] \\
&\propto \mathbb{E}_{q_{-i}}[2y_{ij}\boldsymbol{u}_i^\top \boldsymbol{v}_j] - \mathbb{E}_{q_{-i}}\left[\boldsymbol{u}_i^\top [\boldsymbol{v}_j \boldsymbol{v}_j^\top] \boldsymbol{u}_i\right] - \mathbb{E}_{q_{-i}}\left[\boldsymbol{u}_i^\top \boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1} \boldsymbol{u}_i\right] \\
&\propto 2\mathbb{E}_q[y_{ij}]\boldsymbol{u}_i^\top \boldsymbol{\mu}_{\boldsymbol{v}_j}' - \boldsymbol{u}_i^\top [\boldsymbol{\mu}_{\boldsymbol{v}_j}' \boldsymbol{\mu}_{\boldsymbol{v}_j}'^\top + \boldsymbol{\Sigma}_{\boldsymbol{v}_j}'] \boldsymbol{u}_i - \boldsymbol{u}_i^\top \boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1} \boldsymbol{u}_i\ .
\end{aligned} \tag{2.33}$$

Here once again the expectations are with respect to all variables except $\boldsymbol{u}_i$. The observations affect the update for $\boldsymbol{u}_i$ via $\mathbb{E}_q[y_{ij}]$. Completing the expression in the

last line to perfect square we find that the posterior distribution for $\boldsymbol{u}_i$ is Gaussian with mean and covariance

$$\begin{aligned} q^\star(\boldsymbol{u}_i) &= N(\boldsymbol{\mu}'_{\boldsymbol{u}_i}, \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}) \\ \boldsymbol{\Sigma}'_{\boldsymbol{u}_i} &= \left(\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1} + (\boldsymbol{\mu}'_{\boldsymbol{v}_j}\boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j} + \boldsymbol{\Sigma}'_{\boldsymbol{v}_j})/\sigma^2\right)^{-1} \\ \boldsymbol{\mu}'_{\boldsymbol{u}_i} &= \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}\left(\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}_i} + \mathbb{E}_q[y_{ij}]\boldsymbol{\mu}'_{\boldsymbol{v}_j}/\sigma^2\right) . \end{aligned} \tag{2.34}$$

Due to symmetry the derivation for $\boldsymbol{v}_j$ mirror $\boldsymbol{u}_i$ and the posterior is simply found by swapping variables

$$\begin{aligned} q^\star(\boldsymbol{v}_j) &= N(\boldsymbol{\mu}'_{\boldsymbol{v}_j}, \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}) \\ \boldsymbol{\Sigma}'_{\boldsymbol{v}_j} &= \left(\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1} + (\boldsymbol{\mu}'_{\boldsymbol{u}_i}\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i} + \boldsymbol{\Sigma}'_{\boldsymbol{u}_i})/\sigma^2\right)^{-1} \\ \boldsymbol{\mu}'_{\boldsymbol{v}_j} &= \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}\left(\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}\boldsymbol{\mu}_{\boldsymbol{v}_j} + \mathbb{E}_q[y_{ij}]\boldsymbol{\mu}'_{\boldsymbol{u}_i}/\sigma^2\right) . \end{aligned} \tag{2.35}$$

While the updates shown here are derived by exponentiating the expectation of the joint distribution, they are also equivalent to optimizing the VLB in alternating fashion. For the ordinal observation model the VLB is given by

$$\begin{aligned} &\mathcal{L}(q(\boldsymbol{u}_i), q(\boldsymbol{v}_j), q(y_{ij})) \\ &\quad= \mathbb{E}_q\left[p(x_{ij}, y_{ij}, \boldsymbol{u}_i, \boldsymbol{v}_j)\right] - \mathbb{E}_q[q(\boldsymbol{u}_i)] - \mathbb{E}_q[q(\boldsymbol{v}_j)] - \mathbb{E}_q[q(y_{ij})] \\ &\quad= \frac{\mathbb{E}[y_{ij}]}{\sigma^2}\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i}\boldsymbol{\mu}'_{\boldsymbol{v}_j} - \frac{1}{2\sigma^2}\operatorname{tr}\left\{(\boldsymbol{\Sigma}'_{\boldsymbol{u}_i} + \boldsymbol{\mu}'_{\boldsymbol{u}_i}\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i})(\boldsymbol{\Sigma}'_{\boldsymbol{v}_j} + \boldsymbol{\mu}'_{\boldsymbol{v}_j}\boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j})\right\} \\ &\qquad- \frac{1}{2}\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i}\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}\boldsymbol{\mu}'_{\boldsymbol{u}_i} + \boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i}\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}\boldsymbol{\mu}_{\boldsymbol{u}_i} + \operatorname{tr}\left\{\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}\boldsymbol{\Sigma}'_{\boldsymbol{u}_i}\right\} + \frac{1}{2}\log|\boldsymbol{\Sigma}'_{\boldsymbol{u}_i}| \\ &\qquad- \frac{1}{2}\boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j}\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}\boldsymbol{\mu}'_{\boldsymbol{v}_j} + \boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j}\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}\boldsymbol{\mu}_{\boldsymbol{v}_j} + \operatorname{tr}\left\{\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}\boldsymbol{\Sigma}'_{\boldsymbol{v}_j}\right\} + \frac{1}{2}\log|\boldsymbol{\Sigma}'_{\boldsymbol{v}_j}| \\ &\qquad+ \log\left[\sqrt{2\pi e}\sigma(\Phi(\beta_{ij}) - \Phi(\alpha_{ij}))\right] + \frac{\alpha\phi(\alpha_{ij}) - \beta\phi(\beta)}{2(\Phi(\alpha_{ij}) - \Phi(\beta_{ij}))} . \end{aligned} \tag{2.36}$$

For the continous observation model we have the simpler form

$$\begin{aligned}
&\mathcal{L}(q(\boldsymbol{u}_i), q(\boldsymbol{v}_j)) \\
&\quad = \mathbb{E}_q\left[p(x_{ij}, \boldsymbol{u}_i, \boldsymbol{v}_j)\right] - \mathbb{E}_q[q(\boldsymbol{u}_i)] - \mathbb{E}_q[q(\boldsymbol{v}_j)] \\
&\quad = \frac{x_{ij}}{\sigma^2}\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i}\boldsymbol{\mu}'_{\boldsymbol{v}_j} - \frac{1}{2\sigma^2}\operatorname{tr}\left\{(\boldsymbol{\Sigma}'_{\boldsymbol{u}_i} + \boldsymbol{\mu}'_{\boldsymbol{u}_i}\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i})(\boldsymbol{\Sigma}'_{\boldsymbol{v}_j} + \boldsymbol{\mu}'_{\boldsymbol{v}_j}\boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j})\right\} \\
&\qquad - \frac{1}{2}\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i}\boldsymbol{\Sigma}^{-1}_{\boldsymbol{u}_i}\boldsymbol{\mu}'_{\boldsymbol{u}_i} + \boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i}\boldsymbol{\Sigma}^{-1}_{\boldsymbol{u}_i}\boldsymbol{\mu}_{\boldsymbol{u}_i} + \operatorname{tr}\left\{\boldsymbol{\Sigma}^{-1}_{\boldsymbol{u}_i}\boldsymbol{\Sigma}'_{\boldsymbol{u}_i}\right\} + \frac{1}{2}\log|\boldsymbol{\Sigma}'_{\boldsymbol{u}_i}| \\
&\qquad - \frac{1}{2}\boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j}\boldsymbol{\Sigma}^{-1}_{\boldsymbol{v}_j}\boldsymbol{\mu}'_{\boldsymbol{v}_j} + \boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j}\boldsymbol{\Sigma}^{-1}_{\boldsymbol{v}_j}\boldsymbol{\mu}_{\boldsymbol{v}_j} + \operatorname{tr}\left\{\boldsymbol{\Sigma}^{-1}_{\boldsymbol{v}_j}\boldsymbol{\Sigma}'_{\boldsymbol{v}_j}\right\} + \frac{1}{2}\log|\boldsymbol{\Sigma}'_{\boldsymbol{v}_j}| \,. \qquad (2.37)
\end{aligned}$$

The updates in closed form so they do not require tuning a learning rate. As I mentioned in the introduction, this is a common theme in Kalman filtering that we do not require a learning rate; learning rate-free inference algorithms will be an important theme throughout this thesis.

2.4.2 Approximating and Inferring the Brownian Motion

The variational inference updates for the latent variables $\boldsymbol{u}_i[t]$, $\boldsymbol{v}_j[t]$ depend on the knowledge of prior at time t, which is parametrized by $(\boldsymbol{\mu}_\bullet[t], \boldsymbol{\Sigma}_\bullet[t])$. For the subsequent derivations we will focus on $\boldsymbol{u}_i$ for concreteness; then the vector-specific drift is $a_{\boldsymbol{u}_i}[t]$. We have mentioned that the stochastic integration in Eq. (2.14) is intractable, for which we introduce a Riemann approximation

$$\int_{t-\Delta^{[t]}_{\boldsymbol{u}_i}}^{t} e^{a_{\boldsymbol{u}_i}[s]}ds \;\approx\; e^{a_{\boldsymbol{u}_i}[t]}\Delta^{[t]}_{\boldsymbol{u}_i} \,. \qquad (2.38)$$

Therefore the integrated drift is approximated constant by its end point. Applying this gives the following generative model

$$\begin{aligned}
\boldsymbol{u}_i[t] &\sim N(\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t - \Delta^{[t]}_{\boldsymbol{u}_i}] \;,\; \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t - \Delta^{[t]}_{\boldsymbol{u}_i}] + e^{a_{\boldsymbol{u}_i}[t]}\Delta^{[t]}_{\boldsymbol{u}_i}\boldsymbol{I}) \\
a_{\boldsymbol{u}_i}[t] &\sim N(a_{\boldsymbol{u}_i}[t - \Delta^{[t]}_{\boldsymbol{u}_i}] \;,\; c\;\Delta^{[t]}_{\boldsymbol{u}_i}) \,. \qquad (2.39)
\end{aligned}$$

That is, with this approximation we first draw the log drift value at time t, $a_{\boldsymbol{u}_i}[t]$, according to its underlying Brownian motion. We then use this constant value to

marginalize $\boldsymbol{u}_i$ in the interval $(t - \Delta_{\boldsymbol{u}_i}^{[t]}, t)$. Clearly, as the intervals between observations become smaller, this approximation becomes better. However, if we add Eq. (2.39) to the joint likelihood model of CKF in Eq. (2.15) or Eq. (2.16) the solutions are no longer in closed form.

To circumvent this issue we will learn a point estimate of $a_{\boldsymbol{u}_i}[t]$. We do so by plugging the prior model of Eq. (2.39) into the VLB, and optimize it with respect to $a_{\boldsymbol{u}_i}[t]$. Since the VLB is a lower bound on the marginal likelihood, the estimate obtained this way can be regarded as Type II maximum likelihood. We first write the relevant portion of VLB

$$\begin{aligned}\mathcal{L}(a_{\boldsymbol{u}_i}[t]) &= \mathbb{E}_q \left[\log p(\boldsymbol{u}_i[t] \mid a_{\boldsymbol{u}_i}[t]) + \log p(a_{\boldsymbol{u}_i}[t])\right] \\ &= -\frac{1}{2}\log|\boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t]| - \frac{1}{2}\mathbb{E}_q\left[\ (\boldsymbol{u}_i[t] - \boldsymbol{\mu}_{\boldsymbol{u}_i}[t])^\top \boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}(\boldsymbol{u}_i[t] - \boldsymbol{\mu}_{\boldsymbol{u}_i}[t])\ \right] \\ &\quad - \frac{1}{2c\Delta_{\boldsymbol{u}_i}^{[t]}}(a_{\boldsymbol{u}_i}[t] - a_{\boldsymbol{u}_i}[t - \Delta_{\boldsymbol{u}_i}^{[t]}])^2\ . \end{aligned} \tag{2.40}$$

Next, we take the expectations and write the dependence of $\boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t]$ on $a_{\boldsymbol{u}_i}[t]$ explicitly, but before doing that we introduce additional notation. Since $\boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t - \Delta_{\boldsymbol{u}_i}^{[t]}]$ is positive semidefinite, it admits an eigendecomposition, which can be written as $\boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t - \Delta_{\boldsymbol{u}_i}^{[t]}] = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$ where $\boldsymbol{Q}$ is an orthonormal matrix and Λ is the diagonal matrix of eigenvalues. We also define $\boldsymbol{\omega} = \boldsymbol{Q}^\top(\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t] - \boldsymbol{\mu}_{\boldsymbol{u}_i}[t])$ and $\boldsymbol{\Omega} = \boldsymbol{Q}^\top\boldsymbol{\Sigma}_{\boldsymbol{u}_i}\boldsymbol{Q}$. We can re-write the VLB

$$\begin{aligned}\mathcal{L}(a_{\boldsymbol{u}_i}[t]) = &-\frac{1}{2}\log|\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + e^{a_{\boldsymbol{u}_i}[t]}\Delta_{\boldsymbol{u}_i}^{[t]}| \\ &- \frac{1}{2}\mathbb{E}_q\left[(\boldsymbol{u}_i[t] - \boldsymbol{\mu}_{\boldsymbol{u}_i}[t])^\top(\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + e^{a_{\boldsymbol{u}_i}[t]}\Delta_{\boldsymbol{u}_i}^{[t]})^{-1}(\boldsymbol{u}_i[t] - \boldsymbol{\mu}_{\boldsymbol{u}_i}[t])\right] \\ &- \frac{1}{2c\Delta_{\boldsymbol{u}_i}^{[t]}}(a_{\boldsymbol{u}_i}[t] - a_{\boldsymbol{u}_i}[t - \Delta_{\boldsymbol{u}_i}^{[t]}])^2 \end{aligned} \tag{2.41}$$

which simplifies as

$$\begin{aligned}\mathcal{L}(a_{\boldsymbol{u}_i}[t]) = &-\frac{1}{2}\sum_{d'=1}^{d}\log(\lambda_{d'} + e^{a_{\boldsymbol{u}_i}[t]}\Delta_{\boldsymbol{u}_i}^{[t]}) - \frac{1}{2}\sum_{d'=1}^{d}\frac{\boldsymbol{\omega}_{d'}^2}{\lambda_{d'} + e^{a_{\boldsymbol{u}_i}[t]}\Delta_{\boldsymbol{u}_i}^{[t]}} \\ &- \frac{1}{2}\sum_{d'=1}^{d}\frac{\boldsymbol{\Omega}_{d',d'}}{\lambda_{d'} + e^{a_{\boldsymbol{u}_i}[t]}\Delta_{\boldsymbol{u}_i}^{[t]}}\ . \end{aligned} \tag{2.42}$$

Defining $\eta_{d',1} = \frac{e^{a_{\boldsymbol{u}_i}[t]}\Delta_{\boldsymbol{u}_i}^{[t]}}{\lambda_{d'}+e^{a_{\boldsymbol{u}_i}[t]}\Delta_{\boldsymbol{u}_i}^{[t]}}$ and $\eta_{d',2} = \frac{\boldsymbol{\omega}_{d'}^2+\boldsymbol{\Omega}_{d',d'}}{\lambda_{d'}+e^{a_{\boldsymbol{u}_i}[t]}\Delta_{\boldsymbol{u}_i}^{[t]}}$ the first and second order derivative of VLB with respect to $a_{\boldsymbol{u}_i}[t]$ is

$$\mathcal{L}'(a_{\boldsymbol{u}_i}[t]) = -\frac{a_{\boldsymbol{u}_i}[t] - a_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}]}{c\Delta_{\boldsymbol{u}_i}^{[t]}} - \frac{1}{2}\sum_{d'=1}^{d}\eta_{d',1}(1-\eta_{d',2})$$

$$\mathcal{L}''(a_{\boldsymbol{u}_i}[t]) = -\frac{1}{c\Delta_{\boldsymbol{u}_i}^{[t]}} - \frac{1}{2}\sum_{d'=1}^{d}\eta_{d',1}(1-\eta_{d',1}) + \frac{1}{2}\sum_{d'=1}^{d}\eta_{d',1}(1-2\eta_{d',1})\eta_{d',2}\ . \tag{2.43}$$

Now consider the Taylor expansion of VLB around around $a_{\boldsymbol{u}}[t-\Delta_{\boldsymbol{u}_i}^{[t]}]$, up to and including the second order term. We have

$$\begin{aligned}\mathcal{L}(a_{\boldsymbol{u}_i}[t]) \approx \mathcal{L}(a_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}]) + \mathcal{L}'(a_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}])(a_{\boldsymbol{u}_i}[t] - a_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}]) \\ +\frac{1}{2}\mathcal{L}''(a_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}])(a_{\boldsymbol{u}_i}[t] - a_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}])^2\end{aligned} \tag{2.44}$$

which is simply a quadratic function. Its optimum is given by

$$a_{\boldsymbol{u}_i}^{\star}[t] = a_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}] - \mathcal{L}''(a_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}])^{-1}\mathcal{L}'(a_{\boldsymbol{u}_i}[t-\Delta_{\boldsymbol{u}_i}^{[t]}])\ . \tag{2.45}$$

At every iteration we use Eq. (2.45) to update our estimate of the Brownian motion drift. This derivation is for a single variable $\boldsymbol{u}_i$ but when multiple $\boldsymbol{u}$'s share the same drift the modification is straightforward. For $\boldsymbol{v}_j$ the update is symmetric which is obtained by swapping symbols. Also note that, since the update in Eq. (2.45) depends on the current posterior, its calculation has to be iterated along with the latent variables.

This completes the inference algorithm for the CKF. In order to predict the value of an unseen pair (i,j) we can use the generative model to get

$$\begin{aligned}p(x_{ij}[t]=k) = \int_{\mathbb{R}^d}\int_{\mathbb{R}^d}\int_{\mathcal{I}_k} N(y_{ij}[t] \mid \boldsymbol{u}_i[t]^\top \boldsymbol{v}_j[t], \sigma^2)\ N(\boldsymbol{u}_i[t] \mid \boldsymbol{\mu}'_{\boldsymbol{u}_i}[t], \boldsymbol{\Sigma}'_{\boldsymbol{u}_j}[t]) \times \\ N(\boldsymbol{v}_j[t] \mid \boldsymbol{\mu}'_{\boldsymbol{v}_j}[t], \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t])\ dy_{ij}[t]\ d\boldsymbol{u}_i[t]\ d\boldsymbol{v}_j[t]\end{aligned} \tag{2.46}$$

for ordinal observations and

$$\begin{aligned}p(x_{ij}[t]=k) = \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} N(k \mid \boldsymbol{u}_i[t]^\top \boldsymbol{v}_j[t], \sigma^2)\ N(\boldsymbol{u}_i[t] \mid \boldsymbol{\mu}'_{\boldsymbol{u}_i}[t], \boldsymbol{\Sigma}'_{\boldsymbol{u}_j}[t]) \times \\ N(\boldsymbol{v}_j[t] \mid \boldsymbol{\mu}'_{\boldsymbol{v}_j}[t], \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t])\ d\boldsymbol{u}_i[t]\ d\boldsymbol{v}_j[t]\end{aligned} \tag{2.47}$$

for continuous observations. However, since both factors need to be integrated, there is no closed form solution. One possible solution is to sample from the posteriors $q(\boldsymbol{u}_i[t])$ and $q(\boldsymbol{v}_j[t])$ and carry out Monte-Carlo integration. However, an even simpler solution is to use $\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t]$ and $\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t]$ as point estimates. Plugging these posterior means to the likelihood models in Section 2.3 we obtain

$$p(x_{ij}[t] = k) = \int_{\mathcal{I}_k} N(y_{ij}[t] \mid \boldsymbol{\mu}'_{\boldsymbol{u}_i}[t]^\top \boldsymbol{\mu}'_{\boldsymbol{v}_j}[t] \ , \ \sigma^2) \tag{2.48}$$

for ordinal observations and

$$p(x_{ij}[t] = k) = N(k \mid \boldsymbol{\mu}'_{\boldsymbol{u}_i}[t]^\top \boldsymbol{\mu}'_{\boldsymbol{v}_j}[t] \ , \ \sigma^2) \tag{2.49}$$

for continuous observations. We can now predict the unseen pairs using these simple formulae. In the next section I give full algorithm recipes for CKF and computational complexity analysis.

2.5 Algorithms and Complexity

2.5.1 CKF with Continuous Observations

Algorithm 2.1 CKF with Continuous Observations

Input: $n = 1, \dots, N$: $\{\boldsymbol{X}[t_n]\}$ (data), $\{\Omega[t_n]\}$ (pairs)

d (rank parameter), σ (noise parameter), c (drift parameter)

I (number of iterations)

Output: $n = 1, \dots, N$, $i = 1, \dots, M$, $j = 1, \dots, N$: $\boldsymbol{u}_i[t_n]$, $\boldsymbol{v}_j[t_n]$, $a_{ij}[t_n]$

Initialize: $i = 1, \dots, M:\ \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_0] \leftarrow \texttt{randn}(d)\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_0] \leftarrow \boldsymbol{I}$

$j = 1, \dots, N:\ \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_0] \leftarrow \texttt{randn}(d)\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_0] \leftarrow \boldsymbol{I}$

for $n = 1, \dots, N$ **do**

$\forall_i \in \Omega_{\boldsymbol{u}} : \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}]\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}] + e^{a_{\boldsymbol{u}_i}[t_n]}\Delta_{\boldsymbol{u}_i}^{[t_n]}\boldsymbol{I}$

$\forall_i \in \Omega_{\boldsymbol{v}} : \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}]\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}] + e^{a_{\boldsymbol{v}_j}[t_n]}\Delta_{\boldsymbol{v}_j}^{[t_n]}\boldsymbol{I}$

for $\texttt{iteration} \in \{1, \dots, I\}$ **do**

$$\forall i \in \Omega_{\boldsymbol{u}} : \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n] \leftarrow \left(\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}[t_n] + \textstyle\sum_{j \in \Omega_{\boldsymbol{u}_i}[t_n]} (\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n]\boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j}[t_n] + \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n])/\sigma^2\right)^{-1}$$

$$\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n]\left(\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}[t_n]\boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n] + \textstyle\sum_{j \in \Omega_{\boldsymbol{u}_i}[t_n]} \mathbb{E}_q[y_{ij}[t_n]]\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n]/\sigma^2\right)$$

$$\forall j \in \Omega_{\boldsymbol{v}} : \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n] \leftarrow \left(\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}[t_n] + \textstyle\sum_{i \in \Omega_{\boldsymbol{v}_j}[t_n]} (\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n]\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i}[t_n] + \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n])/\sigma^2\right)^{-1}$$

$$\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n]\left(\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}[t_n]\boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n] + \textstyle\sum_{i \in \Omega_{\boldsymbol{v}_j}[t_n]} \mathbb{E}_q[y_{ij}[t_n]]\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n]/\sigma^2\right)$$

$$\forall i \in \Omega_{\boldsymbol{u}} : a_{\boldsymbol{u}_i}[t_n] \leftarrow a_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}] - \mathcal{L}''(a_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}])^{-1}\mathcal{L}'(a_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}])$$

$$\boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}] + e^{a_{\boldsymbol{u}_i}[t_n]}\Delta_{\boldsymbol{u}_i}^{[t_n]}\boldsymbol{I}$$

$$\forall j \in \Omega_{\boldsymbol{v}} : a_{\boldsymbol{v}_j}[t_n] \leftarrow a_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}] - \mathcal{L}''(a_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}])^{-1}\mathcal{L}'(a_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}])$$

$$\boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}] + e^{a_{\boldsymbol{v}_j}[t_n]}\Delta_{\boldsymbol{v}_j}^{[t_n]}\boldsymbol{I}$$

end for

end for

2.5.2 CKF with Ordinal Observations

Algorithm 2.2 CKF with Ordinal Observations

Input: $n = 1, \ldots, N$: $\{\boldsymbol{X}[t_n]\}$ (data), $\{\Omega[t_n]\}$ (pairs)

d (rank parameter), σ (noise parameter), c (drift parameter)

I (number of iterations)

Output: $n = 1, \ldots, N$, $i = 1, \ldots, M$, $j = 1, \ldots, N$: $\boldsymbol{u}_i[t_n]$, $\boldsymbol{v}_j[t_n]$, $a_{ij}[t_n]$

Initialize: $i = 1, \ldots, M :: \ \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_0] \leftarrow \texttt{randn}(d)\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_0] \leftarrow \boldsymbol{I}$

$i = 1, \ldots, N :\ \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_0] \leftarrow \texttt{randn}(d)\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_0] \leftarrow \boldsymbol{I}$

for $n = 1, \ldots, N$ **do**

$\forall_i \in \Omega_{\boldsymbol{u}} : \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}]\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}] + e^{a_{\boldsymbol{u}_i}[t_n]}\Delta_{\boldsymbol{u}_i}^{[t_n]}\boldsymbol{I}$

$\forall_i \in \Omega_{\boldsymbol{v}} : \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}]\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}] + e^{a_{\boldsymbol{v}_j}[t_n]}\Delta_{\boldsymbol{v}_j}^{[t_n]}\boldsymbol{I}$

for $\texttt{iteration} \in \{1, \ldots, I\}$ **do**

$\forall (i,j) \in \Omega : \mathbb{E}[y_{ij}[t_n]] = m'_{ij} + \sigma\frac{\phi(\alpha_{ij}) - \phi(\beta_{ij})}{\Phi(\beta_{ij}) - \Phi(\alpha_{ij})}$

$\forall i \in \Omega_{\boldsymbol{u}} : \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n] \leftarrow \left(\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}[t_n] + \sum_{j \in \Omega_{\boldsymbol{u}_i}[t_n]}(\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n]\boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j}[t_n] + \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n])/\sigma^2\right)^{-1}$

$\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n]\left(\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}[t_n]\boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n] + \sum_{j \in \Omega_{\boldsymbol{u}_i}[t_n]}\mathbb{E}_q[y_{ij}[t_n]]\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n]/\sigma^2\right)$

$\forall j \in \Omega_{\boldsymbol{v}} : \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n] \leftarrow \left(\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}[t_n] + \sum_{i \in \Omega_{\boldsymbol{v}_j}[t_n]}(\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n]\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i}[t_n] + \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n])/\sigma^2\right)^{-1}$

$\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n]\left(\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}[t_n]\boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n] + \sum_{i \in \Omega_{\boldsymbol{v}_j}[t_n]}\mathbb{E}_q[y_{ij}[t_n]]\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n]/\sigma^2\right)$

$\forall i \in \Omega_{\boldsymbol{u}} : a_{\boldsymbol{u}_i}[t_n] \leftarrow a_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}] - \mathcal{L}''(a_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}])^{-1}\mathcal{L}'(a_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}])$

$\boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}] + e^{a_{\boldsymbol{u}_i}[t_n]}\Delta_{\boldsymbol{u}_i}^{[t_n]}\boldsymbol{I}$

$\forall j \in \Omega_{\boldsymbol{v}} : a_{\boldsymbol{v}_j}[t_n] \leftarrow a_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}] - \mathcal{L}''(a_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}])^{-1}\mathcal{L}'(a_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}])$

$\boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}] + e^{a_{\boldsymbol{v}_j}[t_n]}\Delta_{\boldsymbol{v}_j}^{[t_n]}\boldsymbol{I}$

end for

end for

2.5.3 CKF with Continuous Observations and Fixed Drift

Algorithm 2.3 CKF with Continuous Observations and Fixed Drift

Input: $n = 1, \ldots, N$: $\{\boldsymbol{X}[t_n]\}$ (data), $\{\Omega[t_n]\}$ (pairs)

d (rank parameter), σ (noise parameter)

I (number of iterations)

$i = 1, \ldots, M : a_{\boldsymbol{u}_i}$ (fixed drift for $\boldsymbol{u}_i$'s)

$j = 1, \ldots, N : a_{\boldsymbol{v}_j}$ (fixed drift for $\boldsymbol{v}_j$'s)

Output: $n = 1, \ldots, N$, $i = 1, \ldots, M$, $j = 1, \ldots, N$: $\boldsymbol{u}_i[t_n]$, $\boldsymbol{v}_j[t_n]$

Initialize: $i = 1, \ldots, M :\ \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_0] \leftarrow \texttt{randn}(d)\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_0] \leftarrow \boldsymbol{I}$

$j = 1, \ldots, N :\ \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_0] \leftarrow \texttt{randn}(d)\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_0] \leftarrow \boldsymbol{I}$

for $n = 1, \ldots, N$ **do**

$\forall_i \in \Omega_{\boldsymbol{u}} : \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}]\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}] + e^{a_{\boldsymbol{u}_i}[t_n]}\Delta_{\boldsymbol{u}_i}^{[t_n]}\boldsymbol{I}$

$\forall_i \in \Omega_{\boldsymbol{v}} : \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}]\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}] + e^{a_{\boldsymbol{v}_j}[t_n]}\Delta_{\boldsymbol{v}_j}^{[t_n]}\boldsymbol{I}$

for $\texttt{iteration} \in \{1, \ldots, I\}$ **do**

$\forall i \in \Omega_{\boldsymbol{u}} : \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n] \leftarrow \left(\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}[t_n] + \sum_{j \in \Omega_{\boldsymbol{u}_i}[t_n]} (\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n]\boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j}[t_n] + \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n])/\sigma^2\right)^{-1}$

$\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n]\left(\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}[t_n]\boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n] + \sum_{j \in \Omega_{\boldsymbol{u}_i}[t_n]} \mathbb{E}_q[y_{ij}[t_n]]\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n]/\sigma^2\right)$

$\forall j \in \Omega_{\boldsymbol{v}} : \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n] \leftarrow \left(\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}[t_n] + \sum_{i \in \Omega_{\boldsymbol{v}_j}[t_n]} (\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n]\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i}[t_n] + \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n])/\sigma^2\right)^{-1}$

$\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n]\left(\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}[t_n]\boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n] + \sum_{i \in \Omega_{\boldsymbol{v}_j}[t_n]} \mathbb{E}_q[y_{ij}[t_n]]\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n]/\sigma^2\right)$

end for

end for

2.5.4 CKF with Ordinal Observations and Fixed Drift

Algorithm 2.4 CKF with Ordinal Observations and Fixed Drift

Input: $n = 1, \ldots, N$: $\{\boldsymbol{X}[t_n]\}$ (data), $\{\Omega[t_n]\}$ (pairs)

d (rank parameter), σ (noise parameter)

I (number of iterations)

$i = 1, \ldots, M : a_{\boldsymbol{u}_i}$ (fixed drift for $\boldsymbol{u}_i$'s)

$j = 1, \ldots, N : a_{\boldsymbol{v}_j}$ (fixed drift for $\boldsymbol{v}_j$'s)

Output: $n = 1, \ldots, N$, $i = 1, \ldots, M$, $j = 1, \ldots, N$: $\boldsymbol{u}_i[t_n]$, $\boldsymbol{v}_j[t_n]$

Initialize: $i = 1, \ldots, M : \ \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_0] \leftarrow \texttt{randn}(d)\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_0] \leftarrow \boldsymbol{I}$

$j = 1, \ldots, N : \ \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_0] \leftarrow \texttt{randn}(d)\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_0] \leftarrow \boldsymbol{I}$

for $n = 1, \ldots, N$ **do**

$\forall_i \in \Omega_{\boldsymbol{u}} : \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}]\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{u}_i}[t_n - \Delta_{\boldsymbol{u}_i}^{[t_n]}] + e^{a_{\boldsymbol{u}_i}[t_n]}\Delta_{\boldsymbol{u}_i}^{[t_n]}\boldsymbol{I}$

$\forall_i \in \Omega_{\boldsymbol{v}} : \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}]\ ,\ \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\Sigma}_{\boldsymbol{v}_j}[t_n - \Delta_{\boldsymbol{v}_j}^{[t_n]}] + e^{a_{\boldsymbol{v}_j}[t_n]}\Delta_{\boldsymbol{v}_j}^{[t_n]}\boldsymbol{I}$

for $\texttt{iteration} \in \{1, \ldots, I\}$ **do**

$\forall (i,j) \in \Omega : \mathbb{E}[y_{ij}[t_n]] = m'_{ij} + \sigma\frac{\phi(\alpha_{ij}) - \phi(\beta_{ij})}{\Phi(\beta_{ij}) - \Phi(\alpha_{ij})}$

$\forall i \in \Omega_{\boldsymbol{u}} : \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n] \leftarrow \left(\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}[t_n] + \sum_{j \in \Omega_{\boldsymbol{u}_i}[t_n]} (\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n]\boldsymbol{\mu}'^{\top}_{\boldsymbol{v}_j}[t_n] + \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n])/\sigma^2\right)^{-1}$

$\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n] \leftarrow \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n]\left(\boldsymbol{\Sigma}_{\boldsymbol{u}_i}^{-1}[t_n]\boldsymbol{\mu}_{\boldsymbol{u}_i}[t_n] + \sum_{j \in \Omega_{\boldsymbol{u}_i}[t_n]} \mathbb{E}_q[y_{ij}[t_n]]\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n]/\sigma^2\right)$

$\forall j \in \Omega_{\boldsymbol{v}} : \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n] \leftarrow \left(\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}[t_n] + \sum_{i \in \Omega_{\boldsymbol{v}_j}[t_n]} (\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n]\boldsymbol{\mu}'^{\top}_{\boldsymbol{u}_i}[t_n] + \boldsymbol{\Sigma}'_{\boldsymbol{u}_i}[t_n])/\sigma^2\right)^{-1}$

$\boldsymbol{\mu}'_{\boldsymbol{v}_j}[t_n] \leftarrow \boldsymbol{\Sigma}'_{\boldsymbol{v}_j}[t_n]\left(\boldsymbol{\Sigma}_{\boldsymbol{v}_j}^{-1}[t_n]\boldsymbol{\mu}_{\boldsymbol{v}_j}[t_n] + \sum_{i \in \Omega_{\boldsymbol{v}_j}[t_n]} \mathbb{E}_q[y_{ij}[t_n]]\boldsymbol{\mu}'_{\boldsymbol{u}_i}[t_n]/\sigma^2\right)$

end for

end for

2.5.5 Complexity Analysis

Above I have shown four different versions of CKF; two per ordinal and continuous observations, and two per adjustable and fixed Brownian motion drifts. First we review the additional variables introduced to obtain the full algorithms. $\boldsymbol{X}[t]$ represents the observation matrix at time t; note that we have a different matrix at each time step, permitting the value of a pair (i, j) to change over time. $\Omega[t]$ is the set of all (i, j) for which there is a corresponding observation at $X[t]$. The sets $\Omega_{\boldsymbol{u}}[t]$ and $\Omega_{\boldsymbol{v}}[t]$ are the sets of all i and j for which $(i, j) \in \Omega[t]$. $\Omega_{\boldsymbol{u}_i}[t]$ is the set of all j for which $(i, j) \in \Omega[t]$ and $\Omega_{\boldsymbol{v}_j}[t]$ is defined similarly. We also use `randn` to denote standard normal draws. This is used for initializing $\boldsymbol{u}_i$'s and $\boldsymbol{v}_j$'s.

The overall computational cost in asymptotic notation is the same for all algorithms as the bottleneck is the variational inference loop. The computation cost at any given step is at most $O(d^3)$. This happens in the eigendecomposition of the prior covariances for $\boldsymbol{u}_i's$ and $\boldsymbol{v}_j's$ in the adjustable drift algorithms, and the inversion for finding the posterior covariances for $\boldsymbol{u}_i$'s and $\boldsymbol{v}_j$'s in all algorithms. Since the eigendecomposition is done only once, the inversions in the loop is the dominating term. Then at any given time the complexity is $O(I(\Omega_{\boldsymbol{u}_i}[t] + \Omega_{\boldsymbol{v}_j}[t])d^3)$. We can obtain a uniform bound based on the sparsity of the matrix. In particular, if the number of entries at any given time is bounded as $(\Omega_{\boldsymbol{u}_i}[t] + \Omega_{\boldsymbol{v}_j}[t]) < \Psi$, the cost over the entire time horizon will be $O(NI\Psi d^3)$. We can also see the benefit of using a low rank factorization as the higher order polynomial term depends only on d.

The variational inference updates give a monotonically nondecreasing sequence of VLB values, therefore monitoring this bound for overfitting prevention is not necessary. This is one advantage of Bayesian graphical models over neural networks. With that said we can still compute the appropriate VLB in Eqs. (2.36) or (2.37). The computation cost of doing so is $O(\Psi d^3)$ per iteration, therefore the asymptotic complexity does not change.

2.6 Experiments

In this section we demonstrate the performance of CKF on two types of datasets. Firstly we consider two large-scale movie rating datasets:

- The Netflix dataset containing 100 million movie ratings from 1999 to 2006. The movies ratings are from 1 to 5 stars and distributed across roughly 17K movies and 480K users.

- The MovieLens dataset containing 10 million movie ratings from 1995 to 2009. The ratings are given on a half star scale from 0.5 to 5 and distributed across 10K movies and 71K users.

For the movie ratings we see that the observations are ordinal. As we will show later an adjustable drift is not necessary for this problem so we will use Algorithm 2.4. The second dataset we consider is a financial time series:

- Stock prices recorded at daily close for 433 companies from the AMEX exchange, 2,774 companies from NASDAQ and 3,273 companies from the NYSE for a total of 6,480 stocks and 39.1 million total measurements from 1962–2014. This data is downloaded from Yahoo Finance.

The stock prices are continuous so we use Algorithm 2.1. The adjustable drifts let us extract stock-specific volatility information from data. Below we discuss the setting of CKF parameters and the results in detail.

2.6.1 Movie Rating Data

We first present results on dynamic modeling of the Netflix and MovieLens datasets. These datasets contain user ratings of movies, which are time stamped. We divide the data into matrices where the rows are users and the columns are movies, using a daily resolution. The ability of our model to exploit dynamic information depends on

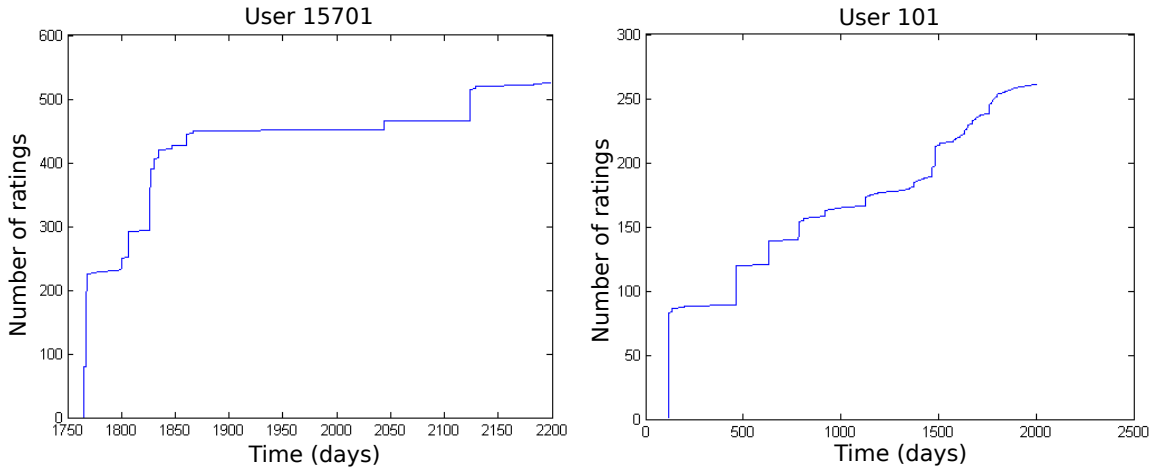


Figure 2.1: Dynamic behavior of two users from the Netflix data set based on the cumulative total of movies rated. Left panel: This user rates large batches of movies in a few sittings. Though the dynamics won't be captured by the model, we still can perform sequential inference for this user to make predictions. Right panel: A more incremental rating pattern that has dynamic value.

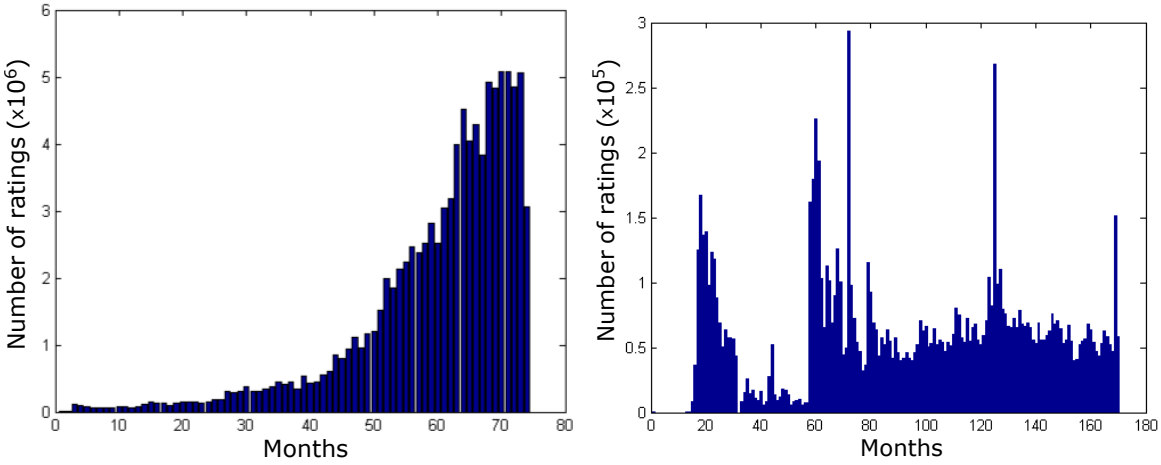
how dynamic the data is. For example, in Figure 2.1 we show the cumulative number of ratings for two users as a function of time. With the first user, we do not expect to have a dynamic benefit as they seem to rate movies at a single time, but we do for the second user. However, as an online algorithm we note that the CKF still can make useful predictions in both cases. Also, in the limiting case that all users rate all movies in a given instant, the CKF will simply reduce to the online zero-drift model. In Figure 2.2 we show the monthly ratings histogram for both data sets, which gives a sense of the dynamic information from the movie perspective.

For comparisons we use the following:

1. CKF: Collaborative Kalman Filter in Algorithm 2.4.

2. Online VB-EM: The non-drift special case of the CKF with $e^{a[t]} = 0$.

3. Batch VB-EM: A variational inference algorithm that uses the ordered probit and prior model of CKF and learns posterior distributions of $\boldsymbol{u}_i$'s, $\boldsymbol{v}_j$'s, and y_{ij}'s



(a) Netflix (b) MovieLens

Figure 2.2: Histograms of the total number of ratings within a month over the course of each data set.

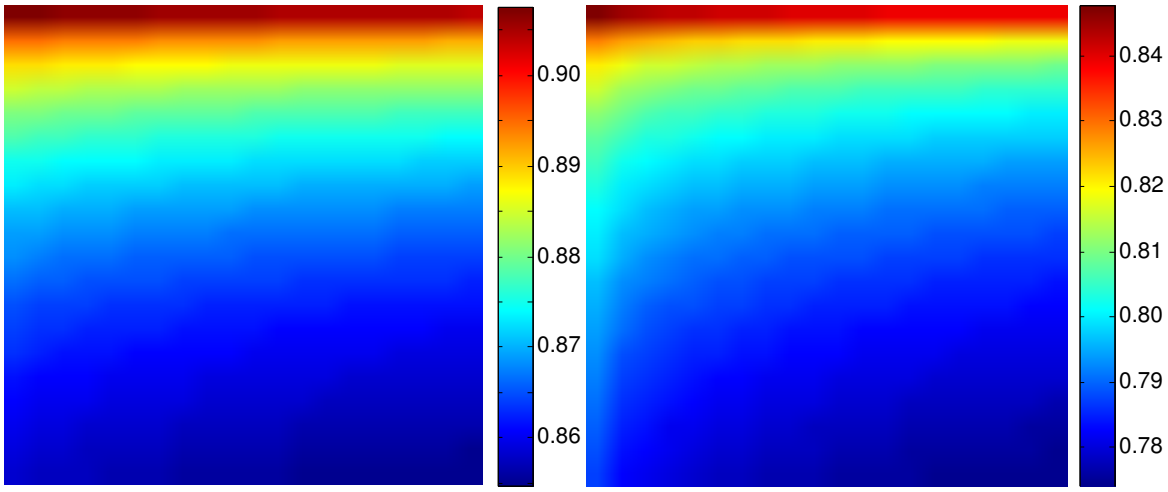
in a batch setting.

4. Batch MAP-EM: A version of probabilistic matrix factorization (PMF) [98] that uses the ordered probit model as above. Point estimates of $\boldsymbol{u}_i$'s and $\boldsymbol{v}_j$'s and the auxiliary variable y_{ij}'s are learned using Expectation Maximization in a batch setting.

5. BPMF: Bayesian PMF [99] without a probit model.

6. M^{3}F: Mixed-membership matrix factorization [79].

Online VB-EM is an online sequential method for Bayesian inference that is similar to other "big data" extensions of the variational inference approaches [19], [55]. The difference between this and the batch model is that we only process each rating once with the online algorithm, while with batch inference we iterate over users and movies several times processing all data in a single iteration, as is more typically done. For batch inference we compare with variational inference (Batch VB-EM) and Expectation Maximization (Batch MAP-EM), both of which use the CKF joint



(a) Netflix (b) MovieLens

Figure 2.3: The RMSE as a function of number of ratings for a user and movie. The (m, n) entry contains the RMSE calculated over user/movie pairs where the user has rated at least $10(m-1)$ movies and the movie has been rated at least $10(n-1)$ times. The value shown in the lower-right corner is 200+ ratings each, which we use in Table 2.1.

likelihood model. Similar to CKF, Batch VB-EM learns a full posterior distribution, whereas Batch MAP-EM finds point estimates according to the MAP rule. BPMF and M^{3}F are also batch inference algorithms, based on Monte Carlo methods.

For parameter selection, we tried several different dimension settings ($d = 10, 20, 30$). We used a fixed drift of $a = 10^{-3}$, which corresponds to setting $c = 0$ since we do not assume there to be fundamental shifts in the overall user or movie landscape. We set σ to be the value that minimizes the KL divergence between the probit and logistic link functions, which we found to be approximately $\sigma = 1.76$. We randomly initialized the mean of the priors on each $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$ at time zero from a standard normal, and set the covariance equal to the identity matrix. For the partition widths, we set $r_k - l_k = \sigma$ for Netflix and $r_k - l_k = \sigma/2$ for MovieLens, which accounts for the half vs. whole star rating system.

We use Root Mean Square Error (RMSE) as the main performance metric, which

Table 2.1: RMSE results for the Netflix 100 million and MovieLens 10 million data sets. Comparisons show an advantage to modeling the dynamic information within the data.

Model	Size	Netflix	MovieLens
CKF	$d = 10$	0.8540	0.7726
	$d = 20$	0.8534	0.7654
	$d = 30$	0.8540	0.7635
Online VB-EM	$d = 10$	0.8682	0.7855
	$d = 20$	0.8707	0.7805
	$d = 30$	0.8668	0.7786
Batch VB-EM	$d = 10$	0.8825	0.7996
	$d = 20$	0.8688	0.7896
	$d = 30$	0.8638	0.7865
Batch MAP-EM	$d = 10$	0.9277	0.9133
	$d = 20$	0.9182	0.9113
	$d = 30$	0.9143	0.9133
BPMF	$d = 30$	0.9047	0.8472
M^3F	$d = 30$	0.9015	0.8447

is shown for all algorithms in Table 2.1. For the batch algorithms, we randomly held out 5% of the data for testing to calculate this value. We ran multiple trials and found that the standard deviation for these algorithms was small enough to omit. We observe that our algorithm outperforms the other baseline algorithms, and so dynamic modeling of user preference does indeed given an improvement in rating prediction. This is especially evident when comparing the CKF with Online VB-EM, the only difference between these algorithms being the introduction of a drift in the state-space vectors.

We also observe the improvement of the variational inference framework in general. For example, by comparing Batch VB-EM with Batch MAP-EM, we see that variational inference provides an improvement over a MAP-EM implementation, which models a point estimate of $\boldsymbol{u}_i[t]$ and $\boldsymbol{v}_j[t]$. Both methods use a latent variable $y_{ij}[t]$ in a probit function for the observed rating, but the variational approach models the uncertainty in these state-space vectors, and so we see that a fully Bayesian approach is helpful. We also found in our experiments that treating the rating $x_{ij}[t]$ as being generated from a probit model and learning a latent $y_{ij}[t]$ is important as well, which we observed by comparing PMF-EM with the original PMF algorithm [98]. We omit these PMF results in the table, which we note gave RMSE results over one as can be seen in the original paper. We also note that the PMF algorithm is the non-dynamic version of [104].

Calculating the RMSE requires a different approach between the online and static models. To calculate the RMSE for the two dynamic models—CKF and the online model—we do not use the test set for the batch models, but instead make predictions of every rating in the data set before using the observed rating to update the model. This gives a more realistic setting for measuring performance of the online models, especially for the CKF where we are interested in predicting the user's rating at that time. Therefore, we must choose which predictions to calculate the RMSE over, since clearly it will be bad for the first several ratings for a particular user or movie. By

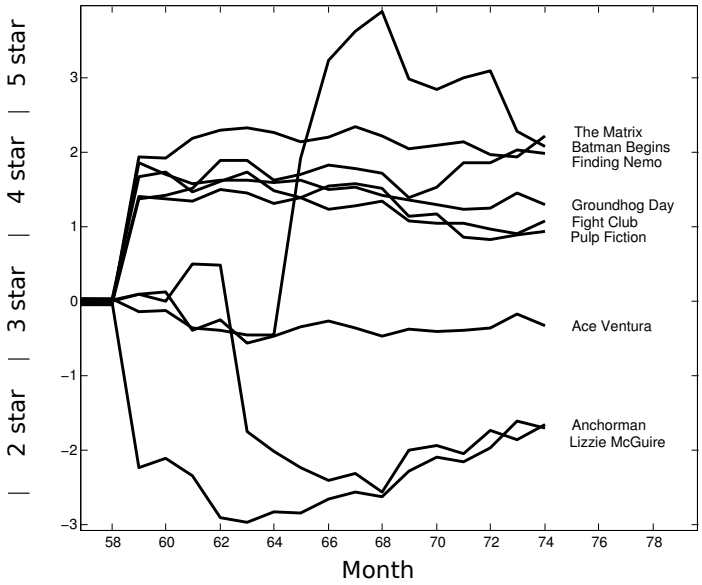


Figure 2.4: An example of a user drift over time as seen through predicted ratings for several movies from the Netflix data set. The y-axis is the latent variable space, which we partition according to star rating as indicated.

looking at the users and movies that appear in the testing sets of the batch models we found that for Netflix each user in the test set had an average of 613 ratings in the training set and each movie in the test set had 53,395 ratings in the training set. For MovieLens this was 447 per user and 7,157 per movie, meaning that in both cases a substantial amount of data was used to learn locations in training before making predictions in testing. Therefore, to calculate our RMSE of the online models we disregard the prediction if either the user or movie has under 200 previous ratings. This arguably still puts the RMSE for our model at a disadvantage, but we noticed that the value did not change much with values larger than 200. We show the RMSE as a function of this number in Figure 2.3.

In Figure 2.4 we show the dynamics of an individual user by plotting the predicted rating for a set of movies as a function of time. We can see an evolving preference in this plot; for example a strong interest in Batman Begins that then slightly decreases with time, while interest in The Matrix begins to increase toward the end. Also, while

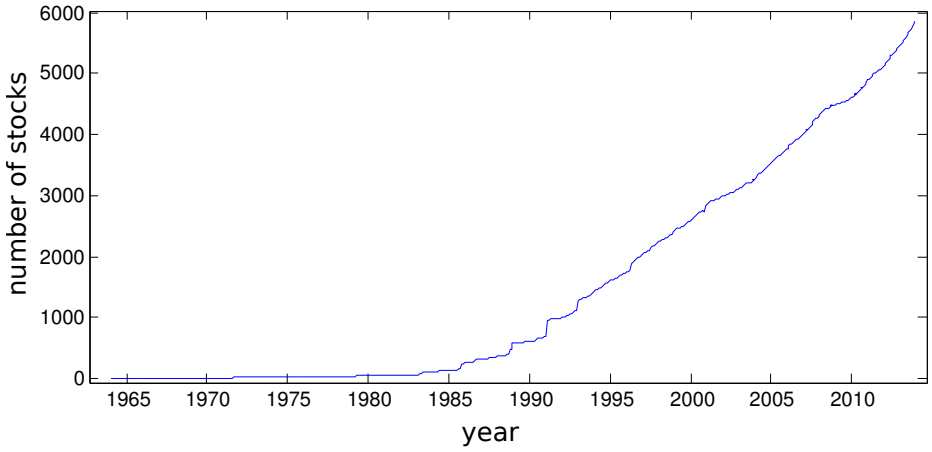


Figure 2.5: The number of actively traded stocks as a function of time.

there is some interest in Anchorman at the beginning, the user then quickly becomes disinterested. In most cases, we observed no clear change in movie preference for a user over time. We consider this to be intuitively reasonable since we argue that few people fundamentally change their taste. However, being able to find those individual users or movies that do undergo a significant change can have an impact on learning the latent vectors for all users and movies since the model is collaborative, and so we argue that modeling time evolution can be valuable even when the actual percentage of dynamically changing behaviors is small.

2.6.2 Stock Price Data

We next present a qualitative evaluation of our model on a stock price dataset. Stock prices are a good example for high dimensional time series where each value gets updated regularly. For our case we use daily closing prices. We also plot the number of active stocks by year in Figure 2.5 where we see that as time goes by, the number of stocks actively being traded increases significantly.

For this problem $x_{ij}[t]$ corresponds to stock prices which is a continuous variable. We can then link it directly fo $\boldsymbol{u}_i[t]$ and $\boldsymbol{v}_j[t]$ using the Gaussian likelihood model of Section 2.3 without using any $y_{ij}[t]$. Also we can learn stock specific volatility

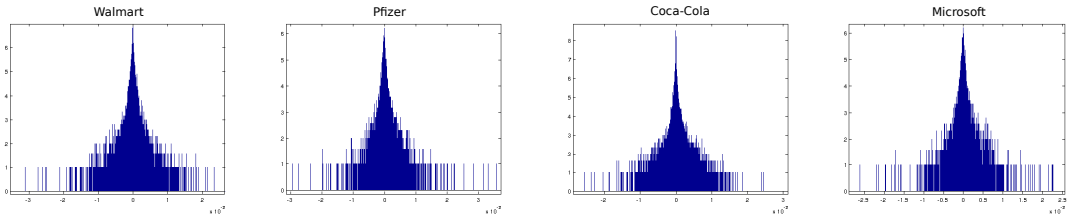
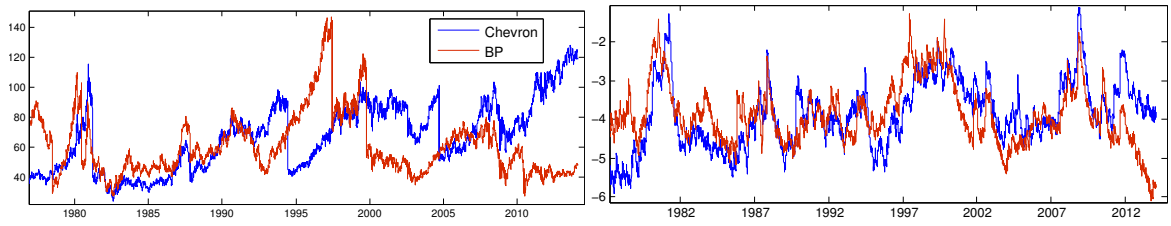


Figure 2.6: A histogram of the tracking errors of the CKF for four stocks using the mean of each q distribution (in $\log_2$ scale). The tracking performance is very accurate using a 5 dimensional latent space (order 10^{-3}). These result are typical of all histograms.

from data using adjustable drifts. Therefore we use Algorithm 2.1. We set the latent dimension $d = 5$, but observed similar results for higher values. We learned stock-specific drift Brownian motions $a_{\boldsymbol{u}_i}[t]$ and set $c = 5 \times 10^{-2}$, which we found to be a good setting through parameter tuning since the learned $a_{\boldsymbol{u}_i}[t]$ were not too smooth, but still stable. We also observe that, for this problem, there is only one state vector corresponding to $\boldsymbol{v}$, which we refer to as a "state-of-the-world" (SOW) vector. For the SOW vector, we use a fixed drift value of $a_{\boldsymbol{v}}[t] = -11.7$ once again corresponds to setting $c = 0$. For the noise standard deviation we set $\sigma = 0.01$, which enforces that the state-space vectors track $x_{ij}[t]$ closely.

We first assess the tracking ability of our model. The ability to accurately track the stock prices indicates that the latent structure being learned is capturing meaningful information about the data set. We show these results as error histograms on $\log_2$ scale in Figure 2.6 for four companies (tracking was very accurate and could not be distinguished visually from the true signal) and mention that these results are representative of all tracking performances. We see from these plots that the prediction errors of the stock prices are small, on the order of 10^{-3}, and so we can conclude that our five dimensional state-space representation for $\boldsymbol{u}_i$'s and $\boldsymbol{v}$ (SOW) is sufficient to capture all degrees of freedom in time across the 6,480 stocks.

We next look more closely at the Brownian motions $a_{\boldsymbol{u}_i}[t]$ learned by the model



(a) Stock price

(b) Brownian motion drift.

Figure 2.7: (a) The historical stock price for two oil companies, BP and Chevron. (b) The log drift Brownian motion ($a_{\boldsymbol{u}_i}[t]$) indicating the volatility of each stock. We see that though the stock prices are different, the volatility of both oil companies share the same shape since they are closely linked in the market.

to capture the volatility of the stock prices. In Figure 2.7(a) we show the historical stock price for two oil companies, BP and Chevron. Below this in Figure 2.7(b) we show the respective functions $a_{\boldsymbol{u}_i}[t]$ learned for these stocks. We see that the volatility of both of these oil companies share similar shapes since they are closely linked in the market. For example in the early 80's, late 90's and late 2000's, oil prices were particularly volatile. This is modeled by the increase of $e^{a_{\boldsymbol{u}_i}[t]}$, which captures the fact that the state vectors are moving around significantly in the latent space during this time to rapidly adjust to stock prices.

We also consider the stock volatility across different market sectors. In Figure 2.8 we show the historical stock prices for five companies along the top row and their respective $a_{\boldsymbol{u}_i}[t]$ below them on the second row. Three of the companies are in the steel market, while the other two are from different markets (pharmaceutics and beverage). We again see that the learned Brownian motion captures a similar volatility for the steel companies. In each case, there is significant volatility associated with the 2008 financial crisis due to the decrease in new construction. This volatility is not found with the pharmaceutics or beverage companies. However, we do see a major spike in volatility for Coca-Cola around 1985, which was a result of their unsuccessful "New Coke" experiment. These observations are confirmed by the respective stock prices.

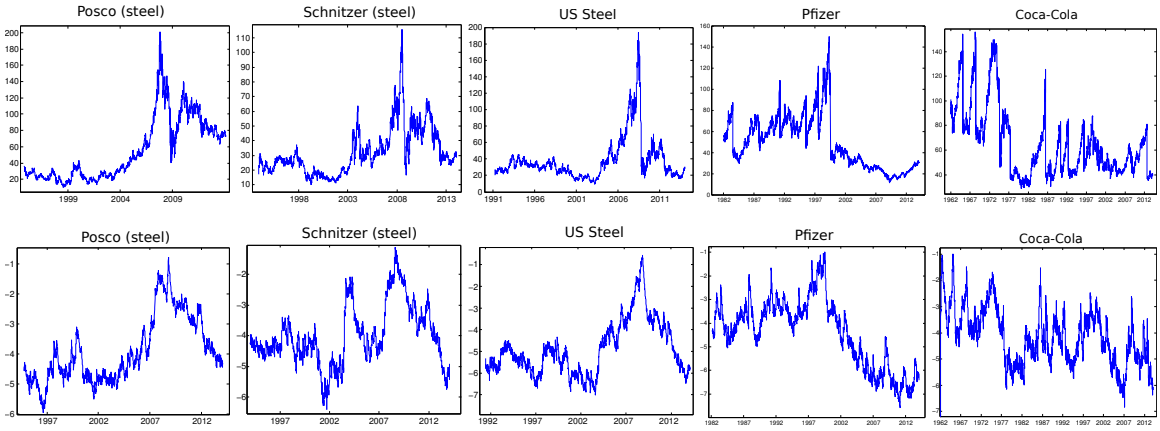


Figure 2.8: (Top row) The historical stock prices for three steel companies, one pharmaceutical and one beverage company. (Bottom row) The corresponding log drift Brownian motions ($a_{\boldsymbol{u}_i}[t]$) for the respective stocks from the top row. We see that the three steel companies shared high volatility during the period of the 2008 financial crisis, but companies in other areas such as the pharmaceutical and beverage industry were not similarly affected.

However, we note that the level of volatility is not only associated with large changes in stock value. In the Pfizer example, we see that the volatility is consistently high for the first half of its stock life, and then decreases significantly for the second half, which is due to the significantly different levels of volatility before and after the year 2000.

2.7 Conclusion

In this chapter we have considered prediction of dyadic time series, which can be represented as a sequence of sparse matrices. The sparse and low-rank structure of the matrices motivate a matrix factorization model for the data generating process. We have focused on extending the matrix factorization to dynamic setting, where the factors are assigned a multidimensional Brownian motion prior. Learning the posterior distributions for the latent factors leads to a nonlinear Kalman filtering

problem which can be efficiently solved with variational inference. The drift parameter of the Brownian motion can also be inferred using the variational lower bound. We showed two kinds of experiments: Firstly we evaluated the model on two movie rating datasets, where the CKF outperforms its competitors. Secondly we have shown qualitative results on stock price data, where stock specific volatility is learned through the Brownian motion drift. The analysis reveals important structure about stocks, such as volatility behavior shared across the stocks in the same sector.

Chapter 3

Dynamic Matrix Factorization for Forecasting

3.1 Introduction

I have introduced Collaborative Kalman Filter (CKF) in Chapter 2. This method relies on matrix factorization and Kalman filtering to model dyadic time series. Dyadic time series are matrix-valued time series where at every time, the interaction of two groups are observed. Using Brownian motion priors I have shown how matrix factorization can naturally be extended to model dyadic time series. Another important property is that, CKF is an online method. So it can process the data in a single stream without storing it in its entirety. This makes CKF suitable for large scale learning problems.

In Chapter 2, we have also focused on analyzing stock prices, for which the observed values are treated as the interaction of individual companies and a state-of-the-world vector. This is a somewhat degenerate case of the CKF model where the observations are not matrix-valued but actually vector-valued. For the stock price dataset, for example, we observe a vector of stock closing prices everyday. The CKF then extracts useful volatility information from this data. While this kind of analysis

is useful, another very important problem in time series analysis is forecasting; here the aim is to predict the future values of the times series from the past. In this chapter we focus on the forecasting problem for vector-valued time series, using matrix factorization.

In the previous chapter I introduced matrix factorization within the context of collaborative filtering and recommender systems [98], [99], [70]. In fact the applications of matrix factorization are more broad and encompass natural language processing [35], [90], image processing [80], finance [6], and power systems analysis [83]. The special subject of non-negative matrix factorization has also received significant attention [71], [72], [119]. However, the application of matrix factorization to forecasting vector-valued time series has been relatively less developed.

To establish the connection, note that a multivariate time series is a sequence of vectors, and when there are missing values the entire collection can be represented by a sparse matrix, for which low-rank representations can be useful. To this end, [116] proposed a temporal regularized matrix factorization based on this observation. A key property of their solution is that the columns of one of the factor matrices is regularized by an AR process. The coefficients of this process are learned from the data, and can be used to forecast future values. With that said, the emphasis in [116] was on batch learning, which for many practical applications might be impractical. For this reason, the focus of this chapter is on online dynamic matrix factorization for forecasting, in a similar spirit to CKF.

Online forecasting is an active area of research [2], [78], [69]. In particular, the recent work [3] considers online predictions with missing values. However, online forecasting of time series has not been considered from a matrix factorization perspective, which we develop here. A key observation in previous works of [2], [3] is that the textbook methods for time series analysis typically assume stationarity and/or Gaussianity of noise, which is often unrealistic; so in this chapter we make fewer assumptions about the data generating process.

To apply dynamic matrix factorization for online forecasting, we will once again use stochastic process priors on the factors. In particular we will use a vector autoregressive (VAR) process on the factor representing a low dimensional time series which is inspired by [116]. The overall model is once again a state-space model. Several other works have also considered dynamic state-space models, but in the batch setting [103] [86], [7], [114]. It is also worthwhile to note that many well established algorithms such as Probabilistic Matrix Factorization [98], Grassmanian Robust Adaptive Subspace Tracking [52], Recursive Projected Compressive Sensing [47], and Online Stochastic Robust PCA [36] can also be used for the online forecasting problem. All these aforementioned techniques leverage the sparse structure of the time series, however the lack of the VAR process in their formulation significantly limits their forecasting ability. This is a gap we aim at filling in this chapter.

Before moving into the details, we also note that there is a significant body of work that considers the missing value and forecasting problems in other settings: [49] proposes a convex optimization framework for transition matrix estimation in vector-valued time series; [33] employs a time-series model to represent missing observations; [101] handles them with an EM algorithm; [28] uses an AR process to impute missing values, and [102] considers the Kalman filtering problem with intermittent observations. The main benefit of using matrix factorization is that, the low rank can be a good choice for the time series considered. And as shown here, non-trivial low rank factorizations can also be learned efficiently in the online setting.

We organize this chapter as follows: Section 3.2 establishes the background for VAR processes and the matrix factorization approach to time series analysis. Section 3.3 is concerned with introducing matrix factorization methods, which finds low-rank factorizations suitable for forecasting. Building on such factorization, Section 3.4 shows how the coefficients of the AR process can be estimated in an optimal manner. Section 3.5 lists algorithms and complexity. Section 3.6 contains experiments with two real datasets with tens of millions of measurements; our experiments show that

the proposed techniques are effective in practical situations. We conclude in Section 3.7.

3.2 Background and Motivation

This section provides background on time series and matrix factorization, and introduces a generative model which we subsequently develop. In this chapter, we are interested in forecasting the future values of a high dimensional time series $\{\boldsymbol{x}_t\}_{t=1}^T$, where each $\boldsymbol{x}_t$ is an $M \times 1$ vector. At each time step t the value of $\boldsymbol{x}_t$ must be predicted before it is observed, denoted by $\widehat{\boldsymbol{x}}_t$, and after observation the model is updated according to a loss function. In this chapter we let the time indices be discrete and equally spaced in time. Total number of samples is T and $[T] = \{1, 2, \ldots, T\}$.

In the well-known Box-Jenkins approach [18], given samples one constructs a signal model by finding (i) a trend, (ii) a seasonal component, and (iii) a noise component, where the latter is typically modeled by an autoregressive moving average (ARMA) model, which is a combination of the AR and MA models. The learned model can then be evaluated using appropriate statistical tests. One drawback of this approach is that finding a trend and seasonal component requires storing and processing the entire data, which might be unsuitable due to storage or computation time requirements. In addition, this methodology is also unsuitable for streaming data.

For these reasons, we start from a generic vector AR process model, VAR(P), of form

$$\boldsymbol{x}_t = \theta_1 \boldsymbol{x}_{t-1} + \ldots + \theta_P \boldsymbol{x}_{t-P} + \boldsymbol{\eta}_{\boldsymbol{x},t} \ , \tag{3.1}$$

where $\boldsymbol{\eta}_{\boldsymbol{x},t}$ is zero mean white noise and P denotes the model order. The choice of P has a major impact on the accuracy of the model, as it captures the maximal lag for correlation. Let the parameters of this model be denoted by $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_P]^\top$. It is clear that, with scalar coefficients VAR(P) corresponds to M copies of an AR(P) model. In addition, when the polynomial $\psi^P - \theta_1 \psi^{P-1} - \ldots - \theta_P$ has roots inside the

unit circle, the model is stationary [48]. For a given finite number of measurements, the parameters of the AR(P) model can be estimated by minimizing the mean square error

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}'} \mathbb{E}_{\boldsymbol{\theta}} \, \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2^2 \, , \tag{3.2}$$

where expectation is taken with respect to the prior $p(\boldsymbol{\theta})$. An advantage of this is, that the optimum linear minimum mean squared error estimator (LMMSE) does not make any distribution assumptions on $p(\boldsymbol{\theta})$ or $p(\boldsymbol{\eta})$, and can be calculated in closed form given the first and second order statistics. Also, unlike least squares or the best linear unbiased estimator, LMMSE is guaranteed to exist.

Returning to matrix factorization, we first observe that a time series can be represented by an $M \times T$ matrix $\boldsymbol{X}$. If d denotes the rank of this matrix, then it is possible to find a $d \times M$ matrix $\boldsymbol{U}$ and a $d \times T$ matrix $\boldsymbol{V}$ such that $\boldsymbol{X} = \boldsymbol{U}^\top \boldsymbol{V}$ [41]. Note that such a factorization is not-unique and there are multiple ways to find it such as singular value decomposition (SVD). Furthermore, when $\boldsymbol{X}$ represents a time series, such a factorization can be interpreted as follows: Since the matrix $\boldsymbol{V}$ is $d \times T$, it corresponds to a compression of the original $M \times T$ matrix $\boldsymbol{X}$. Therefore the matrix $\boldsymbol{V}$ is itself a time series, while the matrix $\boldsymbol{U}$ provides the combination coefficients to reconstruct $\boldsymbol{X}$ from $\boldsymbol{V}$. Based on this observation, [116] proposed a temporal regularized matrix factorization where the regularizer on the columns of $\boldsymbol{V}$ is in the form of an AR process. They showed that such a regularization has notable impact on performance.

Motivated by this, our goal is to learn the factorizations $\boldsymbol{U}$ and $\boldsymbol{V}$, along with the AR model of Eq. (3.2) in the online setting, where at each time instance we observe a single column of the data matrix, $\boldsymbol{x}_t$. While this is similar to previous work on online/dynamic matrix factorization [45], [103], one main issue sets it apart. In the previous work, at each time an $M \times N$ matrix is observed with $N \gg 1$, while in our case the observation is simply $M \times 1$. This is illustrated in Figure 3.1; in (a) we show the batch factorization of an $M \times T$ matrix $\boldsymbol{X}$ and (b) is the case where at each time

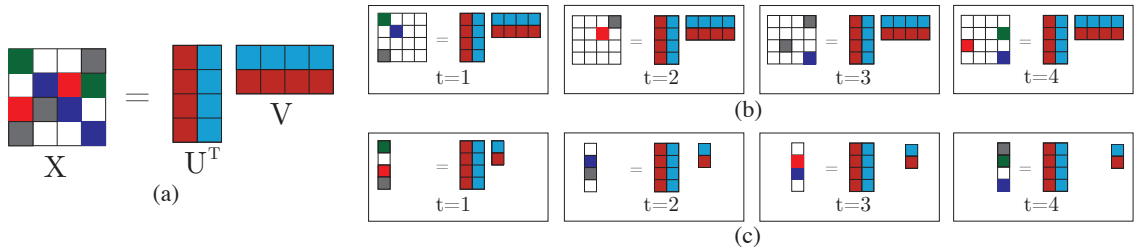


Figure 3.1: Comparison of online matrix factorization schemes. (a) a matrix $\boldsymbol{X}$ is factorized in the batch setting, whereas in (b) at each time a subset of the matrix is observed. For illustrative purposes the observed rank is always greater than one. (c) shows online matrix factorization, where without appropriate regularization (implied in what is shown) the rank cannot exceed one.

a subset of the matrix entries are observed. CKF of Chapter 2 is an example of this. However, when $\boldsymbol{X}$ is a time series matrix, at each time we observe a single column as shown in (c).

A problem with sequentially observing and dynamically factorizing vectors is that the latent rank is at most 1, whereas the batch problem in Figure 3.1(a) will have a solution of rank d. Since the end goal here is to factorize the entire data with two time-varying matrices, it is desirable to start from a rank-d representation and gradually update it. However, finding such factors naively gives poor performance (Figure 4), therefore our task is to devise an effective way of achieving this. This can be done using specific penalties on the matrix $\boldsymbol{U}$, which yields feasible optimization problems, as discussed in the next section. One way to motivate our approach is to consider a probabilistic generative state-space representation for the data, as frequently used in Bayesian methods [13]. The model of CKF is also an example of this. Previous research [7], [5] shows that generative model approach is indeed effective at capturing

temporal dynamics. Our model is

$$\begin{aligned} \boldsymbol{U}_t &= \boldsymbol{U}_{t-1} + \boldsymbol{\eta}_{\boldsymbol{U},t} \\ \boldsymbol{v}_t &= \theta_1 \boldsymbol{v}_{t-1} + \ldots + \theta_P \boldsymbol{v}_{t-P} + \boldsymbol{\eta}_{\boldsymbol{v},t} \\ \boldsymbol{x}_t &= \boldsymbol{U}_t^\top \boldsymbol{v}_t + \boldsymbol{\eta}_{\boldsymbol{x},t} \ , \end{aligned} \tag{3.3}$$

where $[\boldsymbol{\eta}_{\boldsymbol{U},T}]$, $[\boldsymbol{\eta}_{\boldsymbol{v},T}]$, and $[\boldsymbol{\eta}_{\boldsymbol{x},T}]$ are white noise sequences, independent of each other.

For many time series forecasting purposes an AR model is sufficient, as the past values may be the only inputs available. However, if additional predictors are given at any point in time, they can also be incorporated to the generative model. For instance, if an additional set of predictor vectors $\{\boldsymbol{w}_{t,1}, \ldots, \boldsymbol{w}_{t,R}\}$ are provided we can set

$$\boldsymbol{v}_t = \sum_{p=1}^{P} \theta_p \boldsymbol{v}_{t-p} + \sum_{r=1}^{R} \theta'_r \boldsymbol{w}_{t,r} + \boldsymbol{\eta}_{v,t}.$$

In this chapter we assume access only to the time series.

To compare the state-space model in Eq. (3.3) to the related work: If we set $\boldsymbol{v}_t = \boldsymbol{v}_{t-1} + \eta_{\boldsymbol{v},t}$, we recover the setting of matrix factorization. Using the previous value of $\boldsymbol{U}$ as regularizer, we have a feasible optimization problem and the factors can be estimated by either alternating least squares [70] or projected gradients for non-negative factorization [71]. Alternatively, the factors can also be estimated using subspace tracking or robust PCA; these methods bring an additional noise term to the model, which provides outlier robustness. In terms of model complexity and number of parameters, the main difference is that, in the above method we use an AR model for time series embeddings. While this introduces an additional AR order parameter to be set, as we show in the experiments it can lead to significant improvement in forecast accuracy, as multiple past values can be used to predict the future. Without this, the model would predict based only on the current factors. Therefore, when we have a time series where the cross sections are correlated at multiple lags, our generative model can capture the dependencies and provide better forecasts.

3.3 Online Matrix Factorization

We present algorithms that fit an online VAR(P) model to the sequence $[\boldsymbol{v}_T] = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_T\}$. To get a good fit, it is necessary to generate the vectors $[\boldsymbol{v}_T]$ is a proper manner. To illustrate this, for a given measurement vector if we find a factorization $\boldsymbol{x}_t = \boldsymbol{U}_t^\top \boldsymbol{v}_t$, for any orthogonal matrix $\boldsymbol{Q}$ and positive scaling constant a we get $\boldsymbol{x}_t = (a\boldsymbol{Q}\boldsymbol{U}_t)^\top (a^{-1}\boldsymbol{Q}\boldsymbol{v}_t)$. This scaling and rotation could have a significant effect on forecasting accuracy (Figure 3.3). On the other hand, the matrix $\boldsymbol{U}_t$ is a slowly time-varying quantity which means we can constrain its variation. Accurate selection of the penalty on $\boldsymbol{U}_t$ has a dramatic effect on the generated $[\boldsymbol{v}_T]$, which then dictates the forecasting accuracy.

Another assumption we make is that the absolute value of observations are upper bounded by a finite number. Therefore, by scaling we can assume $\sup_{\boldsymbol{x} \in [\boldsymbol{x}_T]} \|\boldsymbol{x}\|_\infty = 1$. For the power data we will consider, this is dictated by the physical constraints of the network; for the traffic data, the measurements are already in percentages.

3.3.1 Fixed Penalty Constraint

The first algorithm we present is based on a simple fixed penalty function on the norms of the factors. The batch version for this algorithm was previously considered in [98]. There, the cost function is

$$f(\boldsymbol{U}, \boldsymbol{v}) = \sum_{m,n} (x_{m,n} - \boldsymbol{u}_m^\top \boldsymbol{v}_n)^2 + \rho_u \sum_m \|\boldsymbol{u}_m\|_2^2 + \sum_n \rho_v \|\boldsymbol{v}_n\|_2^2 \tag{3.4}$$

which is equivalent to adding spherical Gaussian priors to each column of $\boldsymbol{U}$ and $\boldsymbol{v}$. In [98] this is referred to as probabilistic matrix factorization (PMF). This non-convex, unconstrained objective function can be optimized by coordinate descent

$$\begin{aligned} \boldsymbol{u}_m^{(i)} &\leftarrow \Big(\rho_u \boldsymbol{I} + \sum_n \boldsymbol{v}_n^{(i)} \boldsymbol{v}_n^{(i)\top}\Big)^{-1} \Big(\sum_n x_{m,n} \boldsymbol{v}_n^{(i)}\Big) \\ \boldsymbol{v}_n^{(i+1)} &\leftarrow \Big(\rho_v \boldsymbol{I} + \sum_m \boldsymbol{u}_m^{(i)} \boldsymbol{u}_m^{(i)\top}\Big)^{-1} \Big(\sum_m x_{m,n} \boldsymbol{u}_m^{(i)}\Big) \ . \end{aligned} \tag{3.5}$$

Turning to the online case, at each time a single column of $\boldsymbol{X}$ is observed. Using the model of Eq. (3.3), at time t we would like to minimize the following cost function

$$f(\boldsymbol{U}_t, \boldsymbol{v}_t) = \|\boldsymbol{x}_t - \boldsymbol{U}_t^\top \boldsymbol{v}_t\|_2^2 + \rho_u \|\boldsymbol{U}_t - \overline{\boldsymbol{U}}\|_F^2 + \rho_v \|\boldsymbol{v}_t - \overline{\boldsymbol{v}}\|_2^2 \ . \tag{3.6}$$

From the generative model of Eq. (3.3) we see that the predicted values are set as $\overline{\boldsymbol{U}} = \boldsymbol{U}_{t-1}$ and $\overline{\boldsymbol{v}} = \sum_{p=1}^{P} \theta_p \boldsymbol{v}_{t-p}$. In Section 3.4 we discuss how to estimate the parameters in the equation of $\overline{\boldsymbol{v}}$.

Here, $\boldsymbol{U}_t$ is the submatrix of $\boldsymbol{U}$ corresponding to the columns with observation in $\boldsymbol{x}_t$. When there are missing observations, only a subset $\boldsymbol{U}$ gets updated. At a given time t, the number of observations is M_t, and $\boldsymbol{v}_t$ has d parameters. Typically $M_t > d$ and the update for $\boldsymbol{v}_t$ can be feasible even if $\rho_v = 0$; therefore ρ_v is a small set-and-forget constant that we include for numerical stability.[1] On the other hand, $\boldsymbol{U}_t$ contains $M_t d > M_t$ unknowns and the Gram matrix $\boldsymbol{U}_t \boldsymbol{U}_t^\top$ is not invertible. Therefore, $\rho_u > 0$ is necessary to make the problem feasible. Since both ρ_u and ρ_v are fixed at the beginning, we refer to Eq. (3.6) as Fixed Penalty (FP) matrix factorization.

The objective of Eq. (3.6) finds the maximum a posteriori (MAP) solution. Here, we center the priors on the previous value of $\boldsymbol{U}$ and $\overline{\boldsymbol{v}} = \sum_{p=1}^{P} \theta_p \boldsymbol{v}_{t-p}$. Moreover, the ℓ_2-norm terms in Eq. (3.6) suggests that $\boldsymbol{\eta}_{\cdot,t}$ has a density inversely proportional to the distance from the mean. Indeed, this is the only assumption we make about the noise p.d.f. While the most common choice satisfying this requirement would be the Gaussian density; note that its support is the entire $\mathbb{R}^n$ which conflicts with the bounded data assumption. Secondly, from the perspective of Eq. (3.3), the regularization coefficients can be regarded as the inverse noise variance; higher values mean higher trust to the prior and stronger regularization. We note that we will set $\rho_u \gg \rho_v$, which means FP will find a solution for which $\boldsymbol{U}_t$ is close to $\boldsymbol{U}_{t-1}$, i.e., $\boldsymbol{U}_t$ is slowly time-varying. This agrees with the interpretation that, in the batch case $\boldsymbol{U}$ is

[1] Indeed, $\rho_v = 10^{-4}$ for all experiments in this chapter.

Algorithm 3.1 Fixed Penalty Matrix Factorization (FP)

Input: $\boldsymbol{x}_t$, $\mathcal{I}_t$, ρ_u, ρ_v, $\overline{\boldsymbol{U}}$, $\overline{\boldsymbol{v}}$, `max_ite`

Output: $\boldsymbol{U}_t$, $\boldsymbol{v}_t$

Re-assign $\overline{\boldsymbol{U}} \leftarrow \overline{\boldsymbol{U}}(:, \mathcal{I}_t)$

for $i = 1, \ldots,$ `max_ite` **do**

  $\boldsymbol{v}^{(i)} \leftarrow (\rho_v \boldsymbol{I} + \boldsymbol{U}^{(i-1)} \boldsymbol{U}^{(i-1)\top})^{-1} (\rho_v \overline{\boldsymbol{v}} + \boldsymbol{U}^{(i-1)} \boldsymbol{x}_t)$

  $\boldsymbol{U}^{(i)} \leftarrow (\rho_u \boldsymbol{I} + \boldsymbol{v}^{(i)} \boldsymbol{v}^{(i)\top})^{-1} (\rho_u \overline{\boldsymbol{U}} + \boldsymbol{v}^{(i)} \boldsymbol{x}_t^\top)$

end for

Update $\boldsymbol{U}_t(:, \mathcal{I}_t) \leftarrow \boldsymbol{U}$ and overwrite $\boldsymbol{U}_t(:, \mathcal{I}_t^c) \leftarrow \boldsymbol{U}_{t-1}(:, \mathcal{I}_t^c)$.

Update $\boldsymbol{v}_t \leftarrow \boldsymbol{v}$.

Note 1: $\boldsymbol{U}_t(:, \mathcal{I}_t)$ are those columns for which there is a corresponding observation at time t, indexed by $\mathcal{I}_t$. The remaining columns are $\boldsymbol{U}_t(:, \mathcal{I}_t^c)$..

Note 2: At time t, only the observed entries get updated.

a fixed set of coefficients and $\boldsymbol{V}$ contains the compressed time series. Another caution here is that, setting ρ_v too high would over-constrain the problem as both $\boldsymbol{U}_t$ and $\boldsymbol{v}_t$ would be forced to stay close to $\overline{\boldsymbol{U}}$ and $\overline{\boldsymbol{v}}$ while trying to minimize the approximation error to $\boldsymbol{x}_t$.

The update equations for FP are

$$\begin{aligned} \boldsymbol{U}_t^{(i)} &\leftarrow \left(\rho_u \boldsymbol{I} + \boldsymbol{v}_t^{(i)} \boldsymbol{v}_t^{(i)\top}\right)^{-1} \left(\rho_u \overline{\boldsymbol{U}} + \boldsymbol{v}_t^{(i)} \boldsymbol{x}_t^\top\right) \\ \boldsymbol{v}_t^{(i)} &\leftarrow \left(\rho_v \boldsymbol{I} + \boldsymbol{U}_t^{(i)} \boldsymbol{U}_t^{(i)\top}\right)^{-1} \left(\rho_v \overline{\boldsymbol{v}} + \boldsymbol{U}_t^{(i)} \boldsymbol{x}_t\right) . \end{aligned} \tag{3.7}$$

A key argument in Eq. (3.6) is that, the state equations of Eq. (3.3) addresses scaling and rotation issues through $\overline{\boldsymbol{U}}$ and $\overline{\boldsymbol{v}}$. A naive approach, which does not impose any temporal structure on the latent variables, constructs the alternative objective

$$f(\boldsymbol{U}_t, \boldsymbol{v}_t) = \|\boldsymbol{x}_t - \boldsymbol{U}_t^\top \boldsymbol{v}_t\|_2^2 + \rho_u \|\boldsymbol{U}_t\|_F^2 + \rho_v \|\boldsymbol{v}_t\|_2^2 . \tag{3.8}$$

We also consider this alternative "naive" model in the experiments, to show that, in the absence of temporal regularization in Eq. (3.3), scaling and rotation can-

not be prevented,[2] hindering the prediction quality. The FP matrix factorization is summarized in Algorithm 3.1.

3.3.2 Fixed Tolerance Constraint

The fixed penalty approach to matrix factorization suffers from several potential issues. While ρ_v can be set to a small number, setting ρ_u properly has a major impact on performance. It is usually not clear *a priori* which values would yield good results, and oftentimes this may require a large number of cross validations. Another drawback is that ρ_u is fixed for the entire data stream. This may not be desirable as changing the regularization level at different time points may improve performance. For these reasons it can be useful to allow for time varying, self-tunable regularization.

To address this we consider the following problem

$$\begin{aligned} &\min_{\boldsymbol{U}_t, \boldsymbol{v}_t} \|\boldsymbol{U}_t - \overline{\boldsymbol{U}}\|_F^2 + \|\boldsymbol{v}_t - \overline{\boldsymbol{v}}\|_2^2 \\ &\text{s.t.} \quad \|\boldsymbol{x}_t - \boldsymbol{U}_t^\top \boldsymbol{v}_t\|_2^2 \leq \epsilon \end{aligned} \tag{3.9}$$

Instead of having ρ_u and ρ_v, we introduced ϵ. This new parameter forces the approximation error to remain below ϵ. Since this error bound is fixed at the beginning, we call this fixed tolerance (FT) matrix factorization. Here $\overline{\boldsymbol{U}}$ and $\overline{\boldsymbol{v}}$ are defined as in FP. Based on the model in Eq. (3.3), we can interpret the optimization problem of Eq. (3.16) as follows: FT aims finding the point estimates closest to the previous values while keeping the deviation from the likelihood (ML) term at most ϵ. Therefore, while FP is a MAP estimator, FT is a constrained ML estimator.

[2]One alternative way to address this problem would utilize post-processing. In particular, the optimum rotation between two sets of points can be found by solving the Procrustes problem [57]; however this would incur additional computation.

For fixed $\boldsymbol{v}_t$ the problem we would like to solve is

$$\begin{aligned} &\min_{\boldsymbol{U}_t} \|\boldsymbol{U}_t - \overline{\boldsymbol{U}}\|_F^2 \\ &\text{s.t.} \quad \|\boldsymbol{x}_t - \boldsymbol{U}_t^\top \boldsymbol{v}_t\|_2^2 \leq \epsilon \ . \end{aligned} \tag{3.10}$$

The Lagrangian for this problem is

$$\mathcal{L}(\boldsymbol{U}_t, \lambda) = \|\boldsymbol{U}_t - \overline{\boldsymbol{U}}\|_F^2 + \lambda \|\boldsymbol{x}_t - \boldsymbol{U}_t^\top \boldsymbol{v}_t\|_2^2 - \lambda\epsilon \ , \tag{3.11}$$

which yields the following update for $\boldsymbol{U}_t$.

$$\boldsymbol{U}_t \leftarrow (\lambda^{-1}\boldsymbol{I} + \boldsymbol{v}_t\boldsymbol{v}_t^\top)^{-1}(\lambda^{-1}\overline{\boldsymbol{U}} + \boldsymbol{v}_t\boldsymbol{x}_t^\top) \ . \tag{3.12}$$

This is equivalent to (3.7) where $\lambda = \rho_u^{-1}$. But since the Lagrange multiplier changes value with every update of $\boldsymbol{v}_t$ we now have a variable regularizer. The main issue with this FT approach is the structure of the constraint set and its enforcement via the Lagrange multiplier λ. This is a quadratically constrained quadratic program (QCQP), for which there is no closed-form solution in general [16]. The optimization for $\boldsymbol{U}_t$ given $\boldsymbol{v}_t$ can be shown to be convex (Appendix 3.8.1), so off-the-shelf solvers could be employed to find the global optimum. However, using a convex solver at every time step is inefficient and defeats the purpose of scalable online learning.
In fact, a closed form solution to $\boldsymbol{U}_t$ can be found. Defining

$$c_1 = \|\boldsymbol{x}_t - \overline{\boldsymbol{U}}^\top \boldsymbol{v}_t\|_2^2, \quad c_2 = \|\boldsymbol{v}_t\|_2^2, \tag{3.13}$$

and setting the Lagrange multiplier to

$$\lambda^\star = \frac{\sqrt{c_1}}{c_2\sqrt{\epsilon}} - \frac{1}{c_2} \tag{3.14}$$

the optimal update for $\boldsymbol{U}_t$ is

$$\boldsymbol{U}_t \leftarrow (\boldsymbol{I} + \lambda^\star \boldsymbol{v}_t\boldsymbol{v}_t^\top)^{-1}(\overline{\boldsymbol{U}} + \lambda^\star \boldsymbol{v}_t\boldsymbol{x}_t^\top). \tag{3.15}$$

We provide a full derivation of this in Appendix 3.8.1.

Algorithm 3.2 Fixed Tolerance Matrix Factorization (FT)

Input: $\boldsymbol{x}_t$, $\mathcal{I}_t$, ϵ, ρ_v, $\overline{\boldsymbol{U}}$, $\overline{\boldsymbol{v}}$, `max_ite`

Output: $\boldsymbol{U}_t$, $\boldsymbol{v}_t$

Re-assign $\overline{\boldsymbol{U}} \leftarrow \overline{\boldsymbol{U}}(:, \mathcal{I}_t)$

for $i = 1, \ldots,$ `max_ite` **do**

  $\boldsymbol{v}^{(i)} \leftarrow (\rho_v \boldsymbol{I} + \boldsymbol{U}^{(i-1)}\boldsymbol{U}^{(i-1)\top})^{-1}(\rho_v \overline{\boldsymbol{v}} + \boldsymbol{U}^{(i-1)}\boldsymbol{x}_t)$

  Compute c_1, c_2, as in (3.13).

  $\lambda^\star \leftarrow \frac{\sqrt{c_1}}{\sqrt{\epsilon c_2}} - \frac{1}{c_2}$

  $\boldsymbol{U}^{(i)} \leftarrow (\boldsymbol{I} + \lambda^\star \boldsymbol{v}^{(i)}\boldsymbol{v}^{(i)\top})^{-1}(\overline{\boldsymbol{U}} + \lambda^\star \boldsymbol{v}^{(i)}\boldsymbol{x}_t^\top)$

end for

Update $\boldsymbol{U}_t(:, \mathcal{I}_t) \leftarrow \boldsymbol{U}$ and $\boldsymbol{U}_t(:, \mathcal{I}_t^c) \leftarrow \boldsymbol{U}_{t-1}(:, \mathcal{I}_t^c)$.

Update $\boldsymbol{v}_t \leftarrow \boldsymbol{v}$.

Secondly, for a fixed $\boldsymbol{U}_t$ we would like to solve

$$\begin{aligned} &\min_{\boldsymbol{v}_t} \|\boldsymbol{v}_t - \overline{\boldsymbol{v}}\|_2^2 \\ &\text{s.t.} \quad \|\boldsymbol{x}_t - \boldsymbol{U}_t^\top \boldsymbol{v}_t\|_2^2 \leq \epsilon \end{aligned} \tag{3.16}$$

A more challenging issue arises here: For a given threshold ϵ it is not clear if we can find a $\boldsymbol{v}_t$ such that the constraint is satisfied. As an example, when the system of equations is over-determined, the smallest error we can achieve is the least squares error. When the system is underdetermined and the least squares error is greater than ϵ, the value of $\boldsymbol{v}_t$ from the previous iteration will still be the best (Appendix 3.8.2). Unfortunately, the feasible set contains many such *isolated points*. Therefore if we seek the minimum norm solution for $\boldsymbol{v}_t$ in Eq. (3.16) it is likely that the optimization will terminate early, resulting in poor performance.

To fix this we propose the following modification. First note that for a fixed $\boldsymbol{U}_t$ the Lagrangian for $\boldsymbol{v}_t$ is

$$\mathcal{L}(\boldsymbol{v}_t, \lambda) = \|\boldsymbol{v}_t - \overline{\boldsymbol{v}}\|_2^2 + \lambda \|\boldsymbol{x}_t - \boldsymbol{U}_t^\top \boldsymbol{v}_t\|_2^2 - \lambda\epsilon \ , \tag{3.17}$$

and the update is

$$\boldsymbol{v}_t \leftarrow (\lambda^{-1}\boldsymbol{I} + \boldsymbol{U}_t\boldsymbol{U}_t^\top)^{-1}(\lambda^{-1}\overline{\boldsymbol{v}} + \boldsymbol{U}_t\boldsymbol{x}_t). \tag{3.18}$$

Then instead of finding the minimum-norm solution, we can update $\boldsymbol{v}_t$ using the Lagrangian in Eq. (3.17) for a fixed λ. This gives the same update equation as FP, which is given in Eq. (3.7). From the optimization perspective, what we are doing is replacing the least norm update with the ridge regression update (FT vs. FP). The two problems have the same solution when the Lagrange multiplier is the same. Therefore, while the two updates are not equivalent, structurally they are similar to each other. The case when the two updates are the same is established in Appendix 3.8.2. FT is summarized in Algorithm 3.2. The key difference between FT and FP is the computation of λ when updating $\boldsymbol{U}_t$.

3.3.3 Zero Tolerance Constraint

We have discussed two different approaches to online matrix factorization, fixed penalty (FP) and fixed tolerance (FT). From the user perspective, the difference is in replacing one tunable parameter with another. We next discuss a parameter free option in which $\epsilon = 0$, which we refer to as zero tolerance (ZT) matrix factorization. Interpreting from the perspective of the model in Eq. (3.3), ZT estimates the latent factors $\boldsymbol{U}_t$ and $\boldsymbol{v}_t$ that are as close to the prior as possible, while allowing no approximation error on $\boldsymbol{x}_t$; and since no error is allowed, ZT finds a maximum likelihood solution.

The optimization problem now becomes

$$\begin{aligned} &\min_{\boldsymbol{U}_t, \boldsymbol{v}_t} \|\boldsymbol{U}_t - \overline{\boldsymbol{U}}\|_F^2 + \|\boldsymbol{v}_t - \overline{\boldsymbol{v}}\|_2^2 \\ &\text{s.t.} \;\; \boldsymbol{U}_t^\top \boldsymbol{v}_t = \boldsymbol{x}_t \; . \end{aligned} \tag{3.19}$$

This is related to nuclear norm minimization problems [20], [21]. In this scenario, we consider the factored form of the nuclear norm [95] and perform online optimization.

Considering optimizing $\boldsymbol{U}_t$ while $\boldsymbol{v}_t$ is fixed, as before the linear system $\boldsymbol{U}_t\boldsymbol{v}_t = \boldsymbol{x}_t$ is underdetermined for a variable $\boldsymbol{U}_t$. Eq. (3.19) suggests finding the solution with the least Frobenius norm. This generalizes the least norm problem that is considered for linear underdetermined systems to the matrix case. Since the system is underdetermined, the feasible set will contain infinitely many points. (This follows the same reasoning discussed in Appendix 3.8.1.) The optimization can be done with Lagrange multipliers. Following a rescaling, the Lagrangian is given by

$$\mathcal{L}(\boldsymbol{U}_t, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{U}_t - \boldsymbol{U}_{t-1}\|_F^2 + \boldsymbol{\lambda}^\top(\boldsymbol{x}_t - \boldsymbol{U}_t\boldsymbol{v}_t). \tag{3.20}$$

The stationarity conditions are

$$\begin{aligned} \nabla_{\boldsymbol{U}_t}\mathcal{L}(\boldsymbol{U}_t, \boldsymbol{\lambda}) = 0 = \boldsymbol{U}_t - \boldsymbol{U}_{t-1} + \boldsymbol{v}_t\boldsymbol{\lambda}^\top \ , \\ \nabla_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{U}_t, \boldsymbol{\lambda}) = 0 = \boldsymbol{U}_{t-1}^\top\boldsymbol{v}_t - \boldsymbol{x}_t. \end{aligned} \tag{3.21}$$

The solution is then

$$\boldsymbol{\lambda} = \frac{\boldsymbol{U}_{t-1}^\top\boldsymbol{v}_t - \boldsymbol{x}_t}{\boldsymbol{v}_t^\top\boldsymbol{v}_t}, \quad \boldsymbol{U}_t = \boldsymbol{U}_{t-1} - \boldsymbol{v}_t\boldsymbol{\lambda}^\top. \tag{3.22}$$

Here, though $\boldsymbol{\lambda}$ is changing over time, it can no longer be seen as the inverse regularizer of the FP term because ϵ is no longer a tunable parameter, but hard-coded to zero. This is advantageous in that the user does not have to find a good value for it. On the other hand, as we will show in the experiments, the zero tolerance requirement can become too restrictive in some cases, which will then require a higher rank factorization.

The update for $\boldsymbol{v}_t$ suffers from the same problem discussed in Section 3.3.2. For the ZT constraint, consider when $\boldsymbol{U}_t^\top\boldsymbol{v}_t = \boldsymbol{x}_t$ is overdetermined for variable $\boldsymbol{v}_t$. Then the smallest error achievable will be given by the least squares solution, which satisfies $\epsilon_{\text{ls}} > 0$, so the feasible set is empty. However, since the the optimization is done in an alternating manner, this worst case does not occur in practice. In particular, at iteration $i-1$ we have found $(\boldsymbol{U}_{t-1}^{(i-1)}, \boldsymbol{v}_{t-1}^{(i-1)})$ such that $\boldsymbol{U}_{t-1}^{(i-1)\top}\boldsymbol{v}_{t-1}^{(i-1)} = \boldsymbol{x}_t$. This means,

Algorithm 3.3 Zero Tolerance Matrix Factorization (ZT)

Input: $\boldsymbol{x}_t$, $\mathcal{I}_t$, ρ_v, $\overline{\boldsymbol{U}}$, $\overline{\boldsymbol{v}}$, `max_ite`

Output: $\boldsymbol{U}_t$, $\boldsymbol{v}_t$

Re-assign $\overline{\boldsymbol{U}} \leftarrow \overline{\boldsymbol{U}}(:, \mathcal{I}_t)$

for $i = 1, \ldots,$ `max_ite` **do**

  $\boldsymbol{v}^{(i)} \leftarrow (\rho_v \boldsymbol{I} + \boldsymbol{U}^{(i-1)} \boldsymbol{U}^{(i-1)\top})^{-1} (\rho_v \overline{\boldsymbol{v}} + \boldsymbol{U}^{(i-1)} \boldsymbol{x}_t)$

  $\boldsymbol{\lambda} \leftarrow (\overline{\boldsymbol{U}} \boldsymbol{v}^{(i)} - \boldsymbol{x}_t) / (\boldsymbol{v}^{(i)\top} \boldsymbol{v}^{(i)})$

  $\boldsymbol{U}^{(i)} \leftarrow \overline{\boldsymbol{U}} - \boldsymbol{v}^{(i)} \boldsymbol{\lambda}^\top$

end for

Update $\boldsymbol{U}_t(:, \mathcal{I}_t) \leftarrow \boldsymbol{U}$ and $\boldsymbol{U}_t(:, \mathcal{I}_t^c) \leftarrow \boldsymbol{U}_{t-1}(:, \mathcal{I}_t^c)$.

Update $\boldsymbol{v}_t \leftarrow \boldsymbol{v}$.

when we update for $\boldsymbol{v}_t^{(i)}$ for a fixed $\boldsymbol{U}_t^{(i-1)}$, the feasible set will contain at least $\boldsymbol{v}_t^{(i-1)}$. The problem is, if this is the only point contained in the feasible set, the optimization will terminate early. We again address this by replacing the least norm solution with the ℓ_2-regularized one, for which we reintroduce ρ_v as a small parameter. The update is then

$$\boldsymbol{v}_t \leftarrow (\rho_v \boldsymbol{I} + \boldsymbol{U}_t \boldsymbol{U}_t^\top)^{-1} (\rho_v \overline{\boldsymbol{v}} + \boldsymbol{U}_t \boldsymbol{x}_t). \tag{3.23}$$

In summary, ZT is simply the special case of FT where we set $\epsilon = 0$. As ρ_v is a small constant, ZT is effectively a parameter-free matrix factorization method. ZT is summarized in Algorithm 3.3.

3.4 Optimum Sequence Prediction

In Section 3.3 we presented three online matrix factorization approaches with smoothness penalties to constrain the dynamically changing $\boldsymbol{U}$. As discussed in Section 3.2, each column $\boldsymbol{v}_t$ in the product $\boldsymbol{x}_t \approx \boldsymbol{U}_t \boldsymbol{v}_t$ is also generated sequentially. When the columns of the original time series matrix $\boldsymbol{X}$ are correlated, it is natural to model a

correlation structure in $\boldsymbol{V}$. For this reason, we use a VAR(P) model for the columns of $\boldsymbol{V}$

$$\boldsymbol{v}_t = \underbrace{\theta_1 \boldsymbol{v}_{t-1} + \ldots + \theta_P \boldsymbol{v}_{t-P}}_{\overline{\boldsymbol{v}}} + \boldsymbol{\eta}_{\boldsymbol{v},t}. \tag{3.24}$$

The formulae for $\overline{\boldsymbol{v}}$ in Section 3.3 are also based on this.

Therefore, a further task is to find the coefficient vector $\boldsymbol{\theta}$ for this model. While there is no single answer, it is useful to have a flexible estimation method with a small number of assumptions. Thus we adopt the LMMSE estimator since (i) it only needs first and second order statistics, and (ii) optimization is numerically stable, in contrast to, e.g., the best linear unbiased estimator.

We introduce the following notation: First note that (3.24) corresponds to $\boldsymbol{v}_t = \boldsymbol{P}_t\boldsymbol{\theta}$ where $\boldsymbol{P}_t = [\boldsymbol{v}_{t-1} \ \cdots \ \boldsymbol{v}_{t-P}]$ is a $d \times P$ *patch* matrix of the previous P columns. The collection of such matrices is obtained by vertical stacking, $\boldsymbol{P}^\top = [\boldsymbol{P}_1^\top \ \cdots \ \boldsymbol{P}_T^\top]$, which is a $Td \times P$ matrix. Stacking the observation vectors vertically, we obtain $\boldsymbol{p}^\top = [\boldsymbol{v}_1^\top \ \cdots \ \boldsymbol{v}_T^\top]$, a vector with Td elements. The vector $\boldsymbol{\eta}$ is defined similarly.[3]

Using this notation, for the set $[\boldsymbol{v}_T]$, we have the relation

$$\boldsymbol{P}\boldsymbol{\theta} + \boldsymbol{\eta} = \boldsymbol{p} \ , \tag{3.25}$$

which means each vector observation contains information about a latent vector $\boldsymbol{\theta}$. This is different from Kalman Filter [66], where the vector $\boldsymbol{\theta}$ itself is a time-varying latent variable. A good estimator should provide accurate values for $\boldsymbol{\theta}$, with minimal assumptions about the distributions of the random variables involved. To that aim, we let the noise distribution have the first- and second-order statistics

$$\mathbb{E}[\boldsymbol{\eta}_t] = \boldsymbol{0}, \ \mathbb{E}[\boldsymbol{\eta}_{t_1}\boldsymbol{\eta}_{t_2}^\top] = \boldsymbol{\Sigma}_{\boldsymbol{\eta}}\delta(t_1, t_2) \ , \tag{3.26}$$

[3]We observe that at the beginning of Section 3.2 we assumed that the observations start at $T = 1$. To obtain a patch matrix which does not contain any zero-column, we should start constructing matrices $\boldsymbol{P}$ and $\boldsymbol{p}$ from the index $t = P+1$. We omit this detail to simplify the equations. The final algorithm we present, however, addresses this corner case.

where $\delta(t_1, t_2)$ is the Kronecker delta function; this is a white noise process. For the parameter $\boldsymbol{\theta}$ there are two options: (i) it can be treated as an unknown deterministic parameter (classical inference) or (ii) it can be modeled as a random variable (Bayesian inference). LMMSE estimator is based on (ii) so assume that $\boldsymbol{\theta}$ satisfies the following

$$\mathbb{E}[\boldsymbol{\theta}] = \mathbf{0} \ , \ \mathbb{E}[\boldsymbol{\theta}\boldsymbol{\theta}^\top] = \boldsymbol{\Sigma}_{\boldsymbol{\theta}} \ . \tag{3.27}$$

We also note that $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ are assumed independent. Based on the above, we restrict ourselves to the linear estimators of form $\widehat{\boldsymbol{\theta}} = \boldsymbol{W}\boldsymbol{p}$ with $\boldsymbol{W}$ a $dT \times P$ weight matrix. We want to minimize the mean square error

$$\text{MSE} = \mathbb{E}_{\boldsymbol{\theta}} \min_{\widehat{\boldsymbol{\theta}}} \|\boldsymbol{\theta} - \widehat{\boldsymbol{\theta}}\|_2^2 \equiv \mathbb{E}_{\boldsymbol{\theta}} \min_{\boldsymbol{W}} \|\boldsymbol{\theta} - \boldsymbol{W}\boldsymbol{p}\|_2^2, \tag{3.28}$$

for which we can write the expected MSE as

$$\begin{aligned}
\text{MSE}(\boldsymbol{W}) &= \mathbb{E}_{\boldsymbol{\theta}} \ [(\boldsymbol{\theta} - \boldsymbol{W}\boldsymbol{p})^\top (\boldsymbol{\theta} - \boldsymbol{W}\boldsymbol{p})] \\
&= \mathbb{E}_{\boldsymbol{\theta}} \ \text{tr} \ [(\boldsymbol{\theta} - \boldsymbol{W}\boldsymbol{p})(\boldsymbol{\theta} - \boldsymbol{W}\boldsymbol{p})^\top] \\
&= \text{tr} \ \mathbb{E}_{\boldsymbol{\theta}} \ [(\boldsymbol{\theta} - \boldsymbol{W}\boldsymbol{p})(\boldsymbol{\theta} - \boldsymbol{W}\boldsymbol{p})^\top] \\
&= \text{tr} \ \{\boldsymbol{\Sigma}_{\boldsymbol{\theta}} + \boldsymbol{W}\boldsymbol{P}\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{P}^\top\boldsymbol{W}^\top + \boldsymbol{W}\boldsymbol{\Sigma}_{\boldsymbol{\eta}}\boldsymbol{W}^\top - \boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{P}^\top\boldsymbol{W}^\top - \boldsymbol{W}\boldsymbol{P}\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\}
\end{aligned} \tag{3.29}$$

which shows the optimum estimator can be found by matrix differentiation to be

$$\boldsymbol{W} = \boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{P}^\top(\boldsymbol{\Sigma}_{\boldsymbol{\eta}} + \boldsymbol{P}\boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{P}^\top)^{-1}. \tag{3.30}$$

The matrix inversion lemma asserts, for conformable matrices $\boldsymbol{M}_1$, $\boldsymbol{M}_2$, $\boldsymbol{M}_3$, $\boldsymbol{M}_4$

$$[\boldsymbol{M}_1 + \boldsymbol{M}_2\boldsymbol{M}_3\boldsymbol{M}_4]^{-1} = \boldsymbol{M}_1^{-1} - \boldsymbol{M}_1^{-1}\boldsymbol{M}_2[\boldsymbol{M}_3^{-1} + \boldsymbol{M}_4\boldsymbol{M}_1^{-1}\boldsymbol{M}_2]^{-1}\boldsymbol{M}_4\boldsymbol{M}_1^{-1} \tag{3.31}$$

given the inverses exist. Then (3.30) can be re-written as

$$\begin{aligned}
\boldsymbol{W} &= \boldsymbol{\Sigma}_{\boldsymbol{\theta}}\boldsymbol{P}^\top \left[\boldsymbol{\Sigma}_{\boldsymbol{\eta}}^{-1} - \boldsymbol{\Sigma}_{\boldsymbol{\eta}}^{-1}\boldsymbol{P}[\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} + \boldsymbol{P}^\top\boldsymbol{\Sigma}_{\boldsymbol{\eta}}^{-1}\boldsymbol{P}]^{-1}\boldsymbol{P}^\top\boldsymbol{\Sigma}_{\boldsymbol{\eta}}^{-1}\right] \\
&= [\boldsymbol{P}^\top\boldsymbol{\Sigma}_{\boldsymbol{\eta}}^{-1}\boldsymbol{P} + \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}]^{-1}\boldsymbol{P}^\top\boldsymbol{\Sigma}_{\boldsymbol{\eta}}^{-1} \ .
\end{aligned} \tag{3.32}$$

This last line results from algebraic manipulation. The LMMSE estimator is then given by

$$\widehat{\boldsymbol{\theta}} = [\boldsymbol{P}^\top \boldsymbol{\Sigma}_{\boldsymbol{\eta}}^{-1} \boldsymbol{P} + \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}]^{-1} \boldsymbol{P}^\top \boldsymbol{\Sigma}_{\boldsymbol{\eta}}^{-1} \boldsymbol{p} \ . \tag{3.33}$$

It follows from this functional form that the LMMSE estimator reduces to BLUE estimator when a non-informative prior is chosen. This can effectively be written as $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = \infty \boldsymbol{I}$. Furthermore when $\boldsymbol{\Sigma}_{\boldsymbol{\eta}} = \boldsymbol{I}$, BLUE estimator coincides with the LS estimator, yielding the Gauss-Markov theorem [50]. For this paper we consider the case $\boldsymbol{\Sigma}_{\boldsymbol{\eta}} = \boldsymbol{I}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} = r_0 \boldsymbol{I}$ for a tunable parameter r_0.

The transition from (3.30) to (3.32) with $\boldsymbol{\Sigma}_{\boldsymbol{\eta}} = \boldsymbol{I}$ allows us to use matrix partitioning [41] and write (3.33) as

$$\widehat{\boldsymbol{\theta}} = \left[\sum_{t=1}^{T} \boldsymbol{P}_t^\top \boldsymbol{P}_t + \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \right]^{-1} \left[\sum_{t=1}^{T} \boldsymbol{P}_t^\top \boldsymbol{v}_t \right] \ . \tag{3.34}$$

This shows we can compute terms recursively. Specifically, define $\boldsymbol{r}_{l,0} = \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}$ and $\boldsymbol{r}_{r,0} = \boldsymbol{0}$ and the recursions

$$\boldsymbol{r}_{l,t} = \boldsymbol{r}_{l,t-1} + \boldsymbol{P}_t^\top \boldsymbol{P}_t, \ \boldsymbol{r}_{r,t} = \boldsymbol{r}_{r,t-1} + \boldsymbol{P}_t^\top \boldsymbol{v}_t. \tag{3.35}$$

Then, at any given time t have $\widehat{\boldsymbol{\theta}}_t = \boldsymbol{r}_{l,t}^{-1} \boldsymbol{r}_{r,t}$. We now have a fully online algorithm for both factorizing the incoming data matrix and estimating the AR coefficients for the compressed time series. In the next section we show the full algorithms and analyze its complexity.

3.5 Algorithms and Complexity

3.5.1 Fixed Penalty Online Forecasting

Algorithm 3.4 Fixed Penalty Online Forecasting

1: **Input:** $\boldsymbol{X}$ (values), $\{\mathcal{I}_t\}_{t=1}^{T}$ (observations)

2: d (dimension), r_0, ρ_u, ρ_v (regularization), `max_ite`

3: **Output:** $t = 1, \ldots, T$: $\widehat{\boldsymbol{x}}_t$

4: **Initialize:** $\boldsymbol{U}^{(0)} \leftarrow \texttt{rand}(M, d)$, $\boldsymbol{v}^{(0)} \leftarrow \texttt{rand}(d)$.

5: $\boldsymbol{r}_{l,0} \leftarrow r_0 \boldsymbol{I}$, $\boldsymbol{r}_{r,0} \leftarrow \boldsymbol{0}$.

6: **for** $t = 1, \ldots, T$ **do**

7: `// Forecast Step`

8: $\overline{\boldsymbol{U}} \leftarrow \boldsymbol{U}_{t-1}$, $\overline{\boldsymbol{v}} \leftarrow \sum_{l=1}^{P} \theta_l \boldsymbol{v}_{t-l}$ †

9: Forecast: $\widehat{\boldsymbol{x}}_t = \overline{\boldsymbol{U}}^{\top} \overline{\boldsymbol{v}}$

10: `// Fixed Penalty Matrix Factorization`

11: $\boldsymbol{U}_t, \boldsymbol{v}_t \leftarrow \texttt{FP}(\boldsymbol{X}_t,\ \mathcal{I}_t,\ \rho_u,\ \rho_v,\ \overline{\boldsymbol{U}},\ \overline{\boldsymbol{v}},\ \texttt{max\_ite})$

12: `// VAR Parameter Update`

13: **if** $t > P$ **then**

14: $\boldsymbol{P}_t \leftarrow [\boldsymbol{v}_{t-1}, \ldots, \boldsymbol{v}_{t-P}]$

15: $\boldsymbol{r}_{l,t} \leftarrow \boldsymbol{r}_{l,t-1} + \boldsymbol{P}_t^{\top} \boldsymbol{P}_t$

16: $\boldsymbol{r}_{r,t} \leftarrow \boldsymbol{r}_{r,t-1} + \boldsymbol{P}_t^{\top} \boldsymbol{v}_t$

17: $\boldsymbol{\theta}_t \leftarrow \boldsymbol{r}_{l,t}^{-1} \boldsymbol{r}_{r,t}$

18: **end if**

19: **end for**

20: †If $t = 1$ set $\overline{\boldsymbol{U}} = \boldsymbol{0}$ and $\overline{\boldsymbol{v}} = \boldsymbol{0}$. If $1 < t < P$ set $\overline{\boldsymbol{v}} = \boldsymbol{v}_{t-1}$. Otherwise use the update in Line 7.

3.5.2 Fixed Tolerance Online Forecasting

Algorithm 3.5 Fixed Tolerance Online Forecasting

1: **Input:** $\boldsymbol{X}$ (values), $\{\mathcal{I}_t\}_{t=1}^{T}$ (observations)

2: d (dimension), r_0, ϵ, ρ_v (regularization), `max_ite`

3: **Output:** $t = 1, \ldots, T$: $\widehat{\boldsymbol{x}}_t$

4: **Initialize:** $\boldsymbol{U}^{(0)} \leftarrow \texttt{rand}(M, d)$, $\boldsymbol{v}^{(0)} \leftarrow \texttt{rand}(d)$.

5: $\boldsymbol{r}_{l,0} \leftarrow r_0 \boldsymbol{I}$, $\boldsymbol{r}_{r,0} \leftarrow \boldsymbol{0}$.

6: **for** $t = 1, \ldots, T$ **do**

7: `// Forecast Step`

8: $\overline{\boldsymbol{U}} \leftarrow \boldsymbol{U}_{t-1}$, $\overline{\boldsymbol{v}} \leftarrow \sum_{l=1}^{P} \theta_l \boldsymbol{v}_{t-l}$ †

9: Forecast: $\widehat{\boldsymbol{x}}_t = \overline{\boldsymbol{U}}^{\top} \overline{\boldsymbol{v}}$

10: `// Fixed Tolerance Matrix Factorization`

11: $\boldsymbol{U}_t, \boldsymbol{v}_t \leftarrow \texttt{FT}(\boldsymbol{X}_t,\ \mathcal{I}_t,\ \epsilon,\ \rho_v,\ \overline{\boldsymbol{U}},\ \overline{\boldsymbol{v}},\ \texttt{max\_ite})$

12: `// VAR Parameter Update`

13: **if** $t > P$ **then**

14: $\boldsymbol{P}_t \leftarrow [\boldsymbol{v}_{t-1}, \ldots, \boldsymbol{v}_{t-P}]$

15: $\boldsymbol{r}_{l,t} \leftarrow \boldsymbol{r}_{l,t-1} + \boldsymbol{P}_t^{\top} \boldsymbol{P}_t$

16: $\boldsymbol{r}_{r,t} \leftarrow \boldsymbol{r}_{r,t-1} + \boldsymbol{P}_t^{\top} \boldsymbol{v}_t$

17: $\boldsymbol{\theta}_t \leftarrow \boldsymbol{r}_{l,t}^{-1} \boldsymbol{r}_{r,t}$

18: **end if**

19: **end for**

20: †If $t = 1$ set $\overline{\boldsymbol{U}} = \boldsymbol{0}$ and $\overline{\boldsymbol{v}} = \boldsymbol{0}$. If $1 < t < P$ set $\overline{\boldsymbol{v}} = \boldsymbol{v}_{t-1}$. Otherwise use the update in Line 7.

3.5.3 Zero Tolerance Online Forecasting

Algorithm 3.6 Zero Tolerance Online Forecasting

1: **Input:** $\boldsymbol{X}$ (values), $\{\mathcal{I}_t\}_{t=1}^{T}$ (observations)

2: d (dimension), r_0, ρ_v (regularization), `max_ite`

3: **Output:** $t = 1, \ldots, T$: $\widehat{\boldsymbol{x}}_t$

4: **Initialize:** $\boldsymbol{U}^{(0)} \leftarrow \texttt{rand}(M, d)$, $\boldsymbol{v}^{(0)} \leftarrow \texttt{rand}(d)$.

5: $\boldsymbol{r}_{l,0} \leftarrow r_0 \boldsymbol{I}$, $\boldsymbol{r}_{r,0} \leftarrow \boldsymbol{0}$.

6: **for** $t = 1, \ldots, T$ **do**

7: `// Forecast Step`

8: $\overline{\boldsymbol{U}} \leftarrow \boldsymbol{U}_{t-1}$, $\overline{\boldsymbol{v}} \leftarrow \sum_{l=1}^{P} \theta_l \boldsymbol{v}_{t-l}$ †

9: Forecast: $\widehat{\boldsymbol{x}}_t = \overline{\boldsymbol{U}}^{\top} \overline{\boldsymbol{v}}$

10: `// Zero Tolerance Matrix Factorization`

11: $\boldsymbol{U}_t, \boldsymbol{v}_t \leftarrow \texttt{ZT}(\boldsymbol{X}_t,\ \mathcal{I}_t,\ \rho_v,\ \overline{\boldsymbol{U}},\ \overline{\boldsymbol{v}},\ \texttt{max\_ite})$

12: `// VAR Parameter Update`

13: **if** $t > P$ **then**

14: $\boldsymbol{P}_t \leftarrow [\boldsymbol{v}_{t-1}, \ldots, \boldsymbol{v}_{t-P}]$

15: $\boldsymbol{r}_{l,t} \leftarrow \boldsymbol{r}_{l,t-1} + \boldsymbol{P}_t^{\top} \boldsymbol{P}_t$

16: $\boldsymbol{r}_{r,t} \leftarrow \boldsymbol{r}_{r,t-1} + \boldsymbol{P}_t^{\top} \boldsymbol{v}_t$

17: $\boldsymbol{\theta}_t \leftarrow \boldsymbol{r}_{l,t}^{-1} \boldsymbol{r}_{r,t}$

18: **end if**

19: **end for**

20: †If $t = 1$ set $\overline{\boldsymbol{U}} = \boldsymbol{0}$ and $\overline{\boldsymbol{v}} = \boldsymbol{0}$. If $1 < t < P$ set $\overline{\boldsymbol{v}} = \boldsymbol{v}_{t-1}$. Otherwise use the update in Line 7.

3.5.4 Complexity Analysis

We first describe the new notation introduced. $\boldsymbol{X}$ is the entire time series matrix, from which we observe a column $\boldsymbol{x}_t$ at time t. $\mathcal{I}_t$ is the set of observed indices at time t. `rand` is a random draw from the uniform distribution.

The asymptotic complexity of all three algorithms are the same, as they share the same bottlenecks. These are in the matrix factorization and VAR parameter update steps which are iterated for each time step. Since the time series are sparse, suppose that at any given time the highest number of non-zero entries is $s = \sup_t |\mathcal{I}_t|$.

For FP/FT/ZT part of the algorithm the highest computation occurs at matrix multiplications and inversions which are $O(d^3)$ and $O(d^2 s)$. Since $s > d$ typically the cost at this step is $O(d^2 s)$. Since there are `max_ite` iterations—which we will name I_1 for convenience—the overall cost is $O(I_1 d^2 s)$. We can also see the benefit of obtaining closed form solutions for the FT and ZT algorithms. For the FT algorithm, the specific structure of the objective function leads to a QCQP formulation which has a closed form solution. This only requires computing two additional constants, which does not alter the complexity. Similarly, for the ZT algorithm the updates based on Lagrange multipliers do not increase the asymptotic complexity.

For the VAR parameter estimation step, the dominant costs are computing and inverting $\boldsymbol{r}_{l,t}$ which are $O(P^2 d)$ and $O(P^3)$ respectively. Since $P > d$ typically, we can write the cost per time step as $O(P^3)$. The total cost per time step is then $O(I_1 d^2 s + P^3)$ and the cost of running for the entire time series is $O((I_1 d^2 s + P^3)T)$. An important benefit of the low rank approach is that, the complexity is polynomial in d, s, and P, which are typically small constants. So the complexity does not depend on the original time series dimension M, which can be a large number. This is similar to the CKF of Chapter 2, where the complexity is polynomial in the factorization rank and does not depend on the dimensions of the observation matrix.

3.6 Experiments

We test our proposed methodology using two time-series datasets downloaded from the UCI machine learning repository:

- Electricity:[4] Hourly power consumption (megawatts) of 370 customers between Jan. 1, 2012 to Jan. 1, 2015 in Portugal. This gives a matrix of 370 rows and 26,304 columns, with 9,732,480 entries.

- Traffic:[5] Hourly occupancy rates of 963 roads in Bay Area, California, recorded between Jan. 1, 2008 and Mar. 30, 2009. This matrix has 963 rows and 10,560 columns, giving 10,169,280 entries.

For both datasets there are no missing values. We generate missing data in two ways: (i) unstructured sparsity where at each time step the corresponding column of $\boldsymbol{X}$ is uniformly subsampled; (ii) structured sparsity where the sparsity of a row follows a geometric process with certain arrival/departure rates. The forecasting task is to predict the entries at given time step in the future.

We compare with several approaches:

1. Base: This is a base estimator, which estimates the current value as the last observation. If the observation at previous time is missing, then it predicts the average of the last observed vector.

2. AR(P): This is simply the AR model of Eq. (3.1), implemented on the vector observations. We learn the model in an online manner using the LMMSE estimator derived in Section 3.4.

3. PMF: Probabilistic matrix factorization algorithm [98]. To extend PMF to the online setting, the FP cost function in Eq. (3.6) is used, but of course no AR structure is imposed.

[4] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

[5] https://archive.ics.uci.edu/ml/datasets/PEMS- SF

4. CKF: Collaborative Kalman Filter [45]. This is an online approach to matrix factorization problem, where the latent states follow a Brownian motion.

5. ORP: Online robust PCA. The low dimensional time series $\boldsymbol{v}_t$ is estimated by the unconstrained principal component pursuit (PCP) method described in [36].

6. GRA: Grassmannian robust adaptive subspace tracking. This time $\boldsymbol{v}_t$ is estimated by the alternating direction multiplied method (ADMM) approach proposed in [52].

7. NMF: Online non-negative matrix factorization. The implementation is similar to PMF, except that projection steps are added for the updates of $\boldsymbol{U}_t$ and $\boldsymbol{v}_t$ to ensure the factors are nonnegative.

8. Naive MF: This is not a competitive algorithm; it corresponds to the model in (3.8).

9. FP-MF: Fixed penalty matrix factorization (Algorithm 3.4).

10. FT-MF: Fixed tolerance matrix factorization (Algorithm 3.5).

11. ZT-MF: Zero tolerance matrix factorization (Algorithm 3.6).

Among these, the Base and AR(P) are not designed to handle missing or low rank data. Imputation based on the entire data is not possible in an online setting. We found the best-performing approach to be to impute the missing values at time t with the average of observed entries at that time, and use this value as the forecasts for time $t + 1$. While low rank methods are better at handling missing values, as our experiments show, there are also cases where this imputation strategy can be effective (see Figure 3.5(b)).

In terms of the computational complexity, recall that all three of our proposed approaches have $O(I_1 d^2 s + P^3)$ cost per time step. For the others we have: AR(P) is $O(P^2 M)$; PMF, CKF, and NMF are $O(I_1 d^2 s)$; ORP and GRA are $O(I_1 I_2 d^2 s)$. Here,

I_1 is the number of iterations run to update $\boldsymbol{U}_t$ and $\boldsymbol{v}_t$. In addition, for ORP and GRA there is an inner loop for optimizing $\boldsymbol{v}$, using PCP and ADMM respectively, and I_2 denotes the number of times this inner loop is done. Compared to other algorithms, the AR update step we introduced incurs a cost of P^3. Practically, $I_1 \approx P$ and $d^2 \geq P$, therefore the two terms $I_1 d^2 s$ and P^3 are comparable. Consequently, the computational complexity of the three proposed approaches is similar to competing methods. This is a desirable property as we can obtain better forecasts without increasing the complexity. Since both ORP and GRA require an inner loop, when I_2 is comparable to I_1, the I_1^2 term can make these algorithms run slower than the rest. For the AR algorithm, the per time step cost always has the term $M \gg s$ as this scheme is based on filling in the missing values; therefore it will have higher cost than the other algorithms. We use Matlab for implementation on a CPU, and note that the runtime for each algorithm considered is several minutes to process the entire data, for both datasets.

For performance evaluation we use the mean absolute error (MAE) of forecasts one time-step ahead. For a time series with missing observations this is defined as:

$$\epsilon_{\text{MAE}} = \frac{1}{T} \sum_{t=1}^{T} \frac{1}{\ell(\boldsymbol{x}_t)} \|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\|_1 \ , \tag{3.36}$$

where $\boldsymbol{x}_t$ is the observation at time t, $\hat{\boldsymbol{x}}_t$ is its forecast, and $\ell(\boldsymbol{x}_t)$ is the number of observations.

3.6.1 Electricity Data

For this set of experiments, the tunable parameters of each algorithm is set to:

- AR: $P = 24$, $r_0 = 1$
- PMF: $d = 5$, $\rho_u = 1$, $\rho_v = 10^{-4}$
- CKF:[6] $d = 5$, $\nu_d = 10^{-4}$, $\nu_x = 10^{-4}$

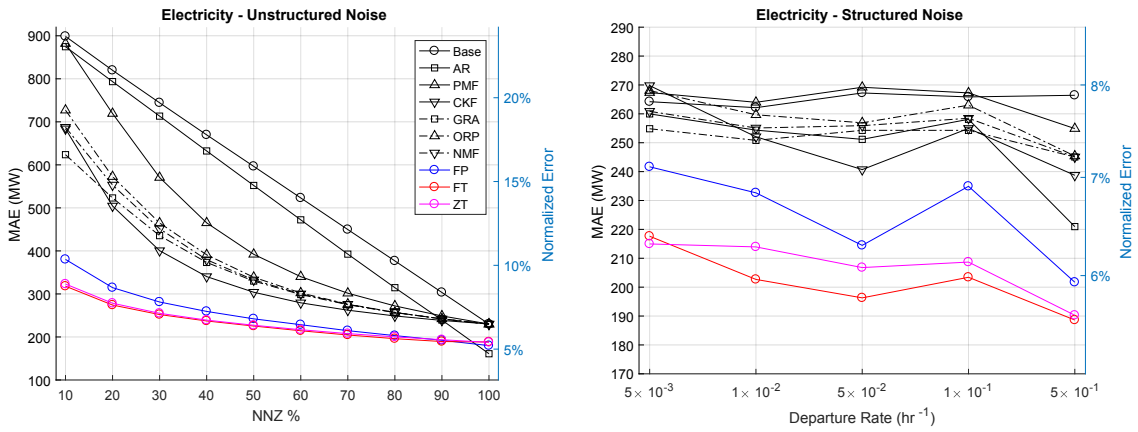
[6] ν_d and ν_x are the drift and measurement noise variance respectively [45].

- Naive MF: $d = 5$, $\rho_u = 1$, $\rho_v = 10^{-4}$
- FP-MF: $d = 5$, $\rho_u = 1$, $\rho_v = 10^{-4}$, $P = 24$, $r_0 = 1$
- FT-MF: $d = 5$, $\epsilon = 0.05$, $\rho_v = 10^{-4}$, $P = 24$, $r_0 = 1$
- LN-MF: $d = 5$, $\rho_v = 10^{-4}$, $P = 24$, $r_0 = 1$
- `max_ite` $= 15$ for all algorithms.

These values were found by cross-validation. We observe that the parameters shared by different algorithms ended up with the same values, which indicates any difference in performance is due to the model structure, rather than parameter settings.

We first show results for one-step ahead prediction when the missingness pattern is unstructured (i.e., totally random). Unstructured sparsity is important in that it makes the learning environment adversarial and algorithms that are unfit for missing values are strongly affected. For our experiments we use 10 different sparsity levels, letting the percentage of observed entries vary from 10% to 100% in 10% increments. We abbreviate this as number of non-zeros (NNZ) as a percentage. For each NNZ level we assess the performance using mean absolute error (MAE) which is in megawatts (MW). We do this for all methods, averaging over 20 sets to ensure statistical significance.

In Figure 3.2(a) we show the one-step ahead prediction performance of all algorithms. When there are no missing observations (NNZ = 100%) the AR model has the best performance, but as NNZ decreases, the performance of AR quickly deteriorates and the three proposed algorithms give the best prediction. This transition already has taken place when NNZ $\leq 90\%$. As the sparsity increases, the base predictor and AR suffer the most. On the other hand, PMF and CKF perform better because they utilize the low-rank representation to impute. Finally, FP/FT/ZT utilize both low-rank and temporal regularization, yielding the best results. Their prediction suffers significantly less than other models as a function of increasing sparsity. FT and ZT



(a) (b)

Figure 3.2: Performance comparison of 10 predictors listed in the beginning of this section, for the electricity dataset. (a) The sparsity pattern is unstructured, and 20 sets of experiments are performed for 10 different levels. (b) The sparsity pattern is structured, and 20 sets of experiments are performed for 5 different departure rates.

perform better than FP, showing that adaptive regularization is indeed useful. We also note that, in general, while most of the online algorithms are concerned with finding factors sequentially, we incorporate an AR process to the generative model in Equation (3.3). This way, while the other online approaches can also find good embeddings sequentially, their predictive power is still limited as these models do not consider dependencies at multiple time lags.

We next experiment with structured sparsity patterns, which is not as adversarial as the previous case. Here, the missing values corresponds to the arrivals of a random process. We use a geometric distribution to generate arrival/departure points for missingness. This sparsity pattern could represent sensor failures or down times. A higher arrival rate indicates increased susceptibility to failure. For the electricity data we set the arrival rate to 0.05 and departure takes values in $\{0.005, 0.01, 0.05, 0.1, 0.5\}$. A higher departure rate means lower sparsity. In Figure 3.2(b) the prediction MAE is shown as a function of departure rates. An immediate observation is that, even if the

sparsity is high (92% when departure rate is 0.005) none of the algorithms deteriorate as much as they do in Figure 3.2(a). Once again, FT has the best performance, and the margin between FT, ZT and FP is more noticeable. On the other hand, the impute-predict scheme of the baseline predictor and the AR predictor also produce acceptable results.

In Section 3.3 we mentioned that rotation and scaling of factor matrices have an important impact on performance. The main reason is, using $\overline{\boldsymbol{U}}$ in regularization encourages smooth variation. The alternative regularization in Eq. (3.8) does not have this feature, as the penalty term on $\boldsymbol{U}_t$ is centered around zero matrix. Since this constraint does not encourage smoothness, it is expected to do worse. We provide evidence for this in Figure 3.3. Here, while the AR model still lets the naive factorization to forecast better than PMF, it is clearly inferior to our methods. This plot also shows that, FP/FT/ZT not only outperform their competitors; but they do so consistently over time. Here once again, mean absolute error (MAE) is computed and plotted over time; in particular, each point in the plot corresponds to a one-week block, which is given by $\epsilon_{\text{MAE}}^{\text{week}} = \frac{1}{T_{\text{week}}} \sum_{t=1}^{T_{\text{week}}} \frac{1}{\ell(\boldsymbol{x}_t)} \|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\|_1$. This plot also gives more information about the electricity data itself. In particular, we observe that all algorithms have higher forecast error during summer times, which indicates electric usage during this season is harder to predict in advance. [7]

Dimensionality and AR order are the two most important parameters which determine how the matrix factorization forecasting performs. We examine performance as a function of these two parameters in Figure 3.4. In Figure 3.4(a) we plot the performance as a function of latent dimensionality for unstructured noise with 80% observed entries. Here we show results for PMF as well as FP, FT, and ZT. First note that a dimension of one gives the worst results for all. This shows the optimum rank is indeed greater than one, and matrix factorization is a suitable approach. The choice $d = 10$ produces best results, although we have used $d = 5$ for our other

[7] This data is collected in Portugal.

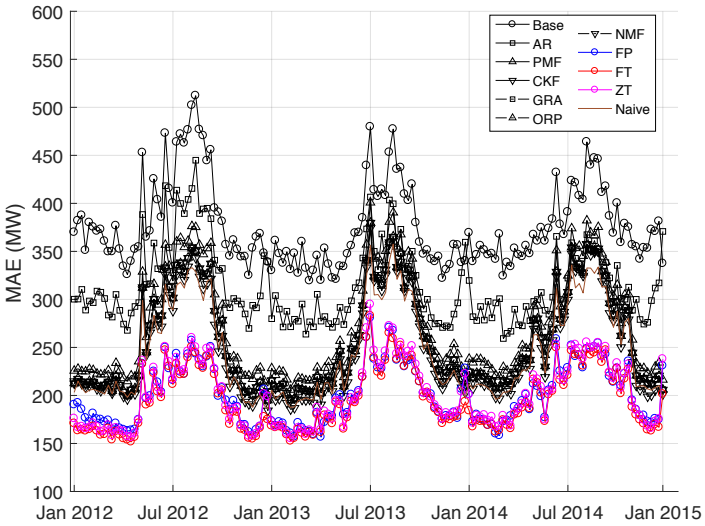
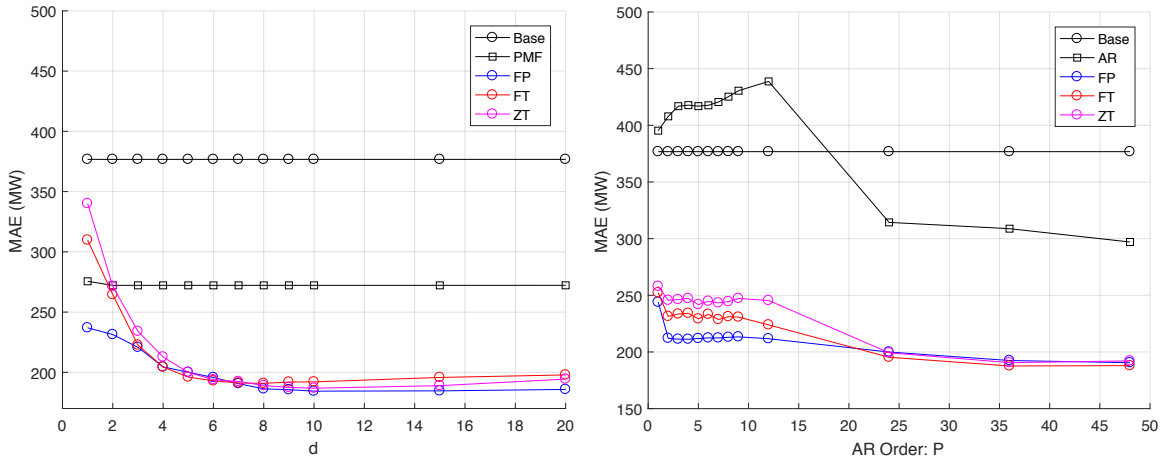


Figure 3.3: Time-varying comparison of all models on electricity data with unstructured sparsity and NNZ = 80%. While naive MF can forecast better than PMF, it is worse than FP/FT/ZT. Overall, FP/FT/ZT consistently outperform the other methods over time, which agrees with the results of Figure 3.2.

experiments, which still produce reliable results with the added benefit of higher compression. When the dimensionality is set low, both FT and ZT perform worse because, as dimension decreases, the fixed or zero tolerance constraint becomes more restrictive, which compromises performance. The degradation for ZT is greater than FT, which is expected since it is a zero error constraint. Therefore, when d needs to be low, we can use FT instead of ZT with $\epsilon > 0$ to provide better factorization as it provides slackness. In Figure 3.4(b) we show results as a function of AR order where $P \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 12, 24, 36, 48\}$. We note a jump in performance as P moves from 12 to 24. This makes sense because $P = 24$ (a 24-hour period) indicates a daily periodicity for power consumption. Another observation is, in the case of missing data, the AR model gives unreliable estimates for lower orders, which suggests a correct choice of model order is important when imputing missing values.



(a) Performance vs. rank

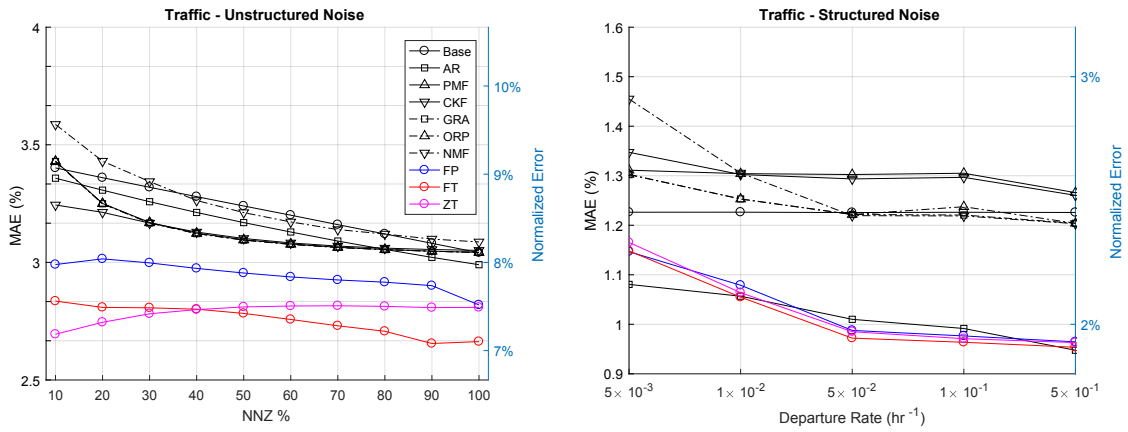
(b) Performance vs. AR order

Figure 3.4: Plot of prediction performances as a function of (a) rank and (b) AR order. Both plots obtained for electricity dataset with unstructured sparsity and NNZ = 80%.

3.6.2 Traffic Data

For the traffic dataset, the parameter settings are:

- AR: $P = 24$, $r_0 = 1$
- PMF: $d = 20$, $\rho_u = 10^{-1}$, $\rho_v = 10^{-4}$
- CKF: $d = 20$, $\nu_d = 10^{-6}$, $\nu_x = 10^{-6}$
- Naive MF: $d = 20$, $\rho_u = 10^{-1}$, $\rho_v = 10^{-4}$
- FP-MF: $d = 20$, $\rho_u = 10^{-1}$, $\rho_v = 10^{-4}$, $P = 24$, $r_0 = 1$
- FT-MF: $d = 20$, $\epsilon = 0.05$, $\rho_v = 10^{-4}$, $P = 24$, $r_0 = 1$
- LN-MF: $d = 20$, $\rho_v = 10^{-4}$, $P = 24$, $r_0 = 1$
- `max_ite` = 15 for all algorithms.

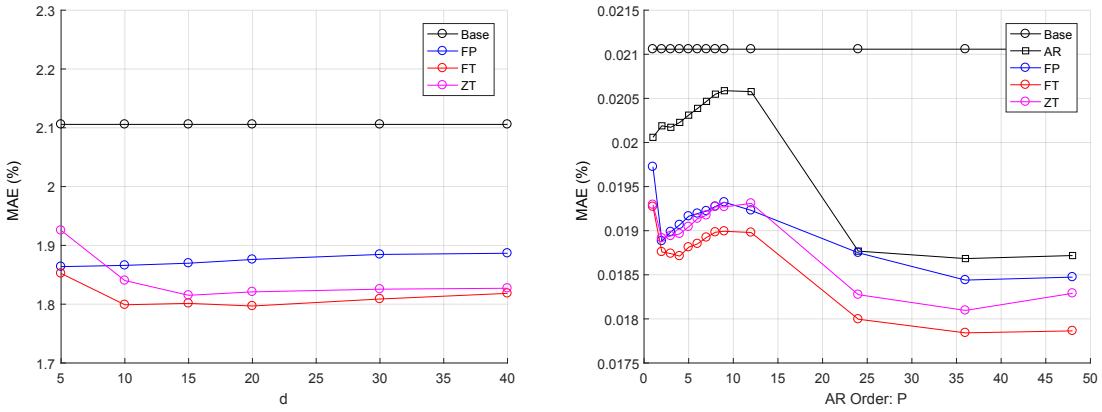


(a) (b)

Figure 3.5: Performance comparison of 10 predictors listed in the beginning of this section, for the traffic dataset. (a) The sparsity pattern is unstructured, and 20 sets of experiments are performed for 10 different levels. (b) The sparsity pattern is structured, and 20 sets of experiments are performed for 5 different departure rates.

In particular, we increase d to 20 to account for the increased dimensionality of the input data.

Once again, we consider structured and unstructured sparsity. The sparsity levels, arrival/departure rates, and the number of test sets are identical to what was used previously. In Figure 3.5(a) the results for unstructured sparsity is shown. In this case, once again the best results are given by the proposed methods. On the other hand, Base and AR do not deteriorate as severely as fo the electricity data in Figure 3.2(a). Also, PMF and CKF are no longer competitive on this data. In Figure 3.5(b) we consider structured sparsity. This case is more unique from those previously considered. First, even for highly sparse inputs the performance of the base estimator does not deteriorate. Since Figure 3.5(a) already shows that the sparsity does not have a very strong effect in adversarial case, the results for structured noise are not surprising. Since this is true for the base predictor, the AR predictor remains competitive as well. In fact, here the fill step is good enough to alleviate the missing



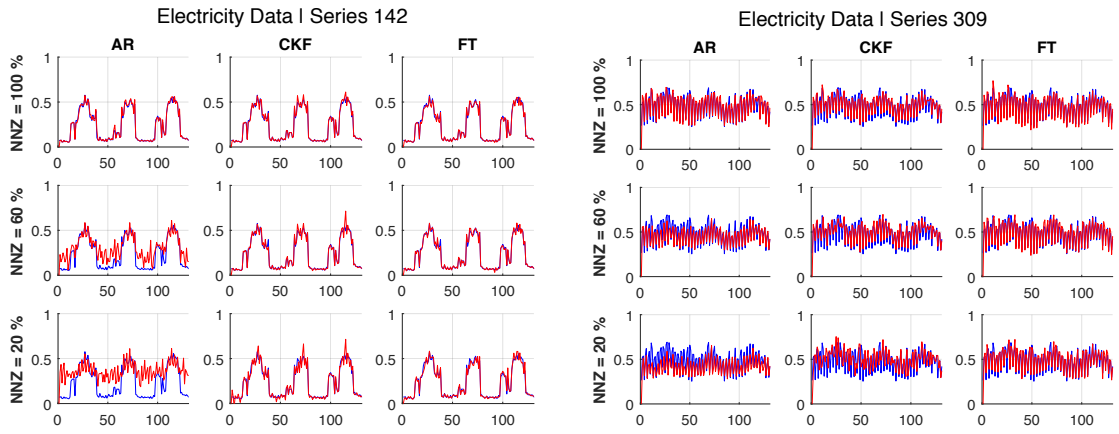
(a) Performance vs. rank

(b) Performance vs. AR order

Figure 3.6: Plot of prediction performances as a function of (a) rank and (b) AR order. Both plots obtained for traffic dataset with unstructured sparsity and NNZ = 50%.

data problem, so even if the data is sparse, the AR predictor can provide accurate forecasts. If we instead filled missing entries with zeros, this apparent advantage of AR disappears. Nevertheless, the difference between AR and FP/FT/ZT is small.

Similar to electricity data, we analyze the prediction performance as a function of rank and AR order in Figure 3.6. In Figure 3.6(a) we consider the effect of latent dimensionality. Unlike the electricity data, choosing $d = 1, 2$ resulted in unstable performance for FT and ZT because for this data choosing such a low rank is inappropriate. We therefore sweep $d \in \{5, 10, 15, 20, 30, 40\}$ and observe once $d \geq 10$ all factorizations produce consistent results. Once again we note that ZT is more susceptible to error compared to FT when dimension is low, as the zero tolerance constraint is more restrictive. In Figure 3.6(b) we show the effect of AR order. Here the results are similar to the electricity data; setting $P = 24$ yields good results for FP, FT, and ZT. Once again, a one-day periodicity is reasonable, since traffic intensity has a daily pattern, e.g. rush hours in the morning and evening.



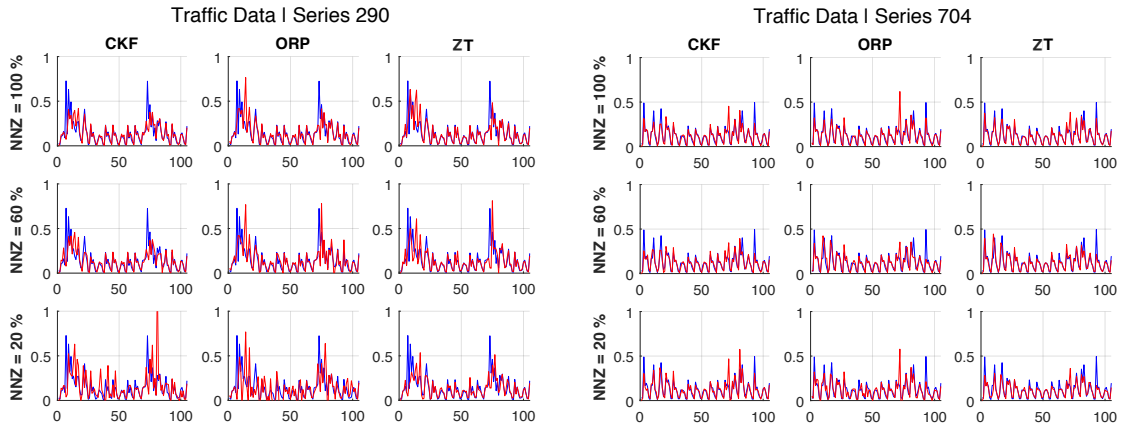
(a) (b)

Figure 3.7: (a) Plot of time series no. 142 of Electricity data (blue) vs. forecasts (red) of AR, CKF, and FT for three sparsity levels, expressed in NNZ percentage. (b) Plot of time series no. 309 of Electricity data (blue) vs. forecasts (red) of AR, CKF, and FT for three sparsity levels, expressed in NNZ percentage.

3.6.3 Forecasts on Individual Time Series

In this section we provide supplemental plots where we show the forecast values compared against the actual time series. Since the original series contains $\sim 10^4$ samples we use a sampling rate of 200 for electricity and 100 for traffic.

In Figure 3.7(a) and 3.7(b) we show the prediction accuracy on two individual time series of the electricity data. These correspond to customers 142 and 309. Each plot is a 3×3 grid. The columns correspond to three filters AR(P), CKF, and FT; and the rows are in increasing (decreasing) sparsity (NNZ percentage). We show normalized plots to make comparisons across different series easier. The results do not change for the unnormalized case; everything is simply multiplied by a constant number. We use unstructured noise as there are bigger differences between filters as a function of NNZ for this case. For series number 142 we see that AR has the worse deterioration, losing track for NNZ $= 20\%$. CKF, being a low-rank method, has good performance for low sparsity as well, but we can see overshooting at peaks for NNZ



(a) (b)

Figure 3.8: (a) Plot of time series no. 290 of Traffic data (blue) vs. forecasts (red) of CKF, ORP, and LN for three sparsity levels, expressed in NNZ percentage. (b) Plot of time series no. 704 of Traffic data (blue) vs. forecasts (red) of CKF, ORP, and LN for three sparsity levels, expressed in NNZ percentage.

= 60% and NNZ = 20%. Finally, FT has good predictions for all sparsity levels, without such overshooting. For series number 309, once again AR deteriorates the most. CKF does better although the deterioration is particularly visible for NNZ = 60% and NNZ = 20%. Again FT has the best performance.

In Figures 3.8(a) and 3.8(b) we show the prediction accuracy on two individual time series of the traffic data. Here individual series correspond to occupancy rate of different roads. Once again unstructured noise is used and we compare CKF, ORP, and ZT. For both plots we see that where CKF and ORP have overshoots or larger errors, ZT performs better. However, compared to electricity data the differences between the three methods are less pronounced, and deterioration as a function of NNZ is also less severe.

3.7 Conclusion

We have considered the problem of forecasting future values of high dimensional time series. A high dimensional time series can be treated as a matrix, where each column denotes realization at a particular time, and when missing values are present, a low rank matrix factorization can be used as a building block for a forecasting that also imputes missing values. Based on this idea, we proposed three methods which can perform matrix factorization in the online setting. These approaches differ by the type of regularization imposed, and a key conclusion is that time-varying regularization can be achieved through a constrained optimization problem.

The matrix factorization component provides a low dimensional representation, which are then used to learn an AR model on next values in this low dimensional space. This in turn forms the basis of forecasting. We derived the optimum LMMSE estimator to find these AR coefficients that only requires the first and second order statistics of the noise terms. Finally we considered two real datasets—electricity and traffic—and showed that when missing values are present in the data, our methods can provide more reliable forecasts. With this we conclude the applications of matrix factorization to time series analysis. In the next chapter I will focus on the more general problem of nonlinear Kalman filtering.

3.8 Appendix to Chapter 3

3.8.1 Fixed tolerance update: $\boldsymbol{U}_t$

At any given iteration-i of the E-step in Algorithm 3.2, for a fixed $\boldsymbol{v}_t^{(i)}$, we want to find

$$\boldsymbol{U}_t^{(i)} = \arg\min_{\boldsymbol{U}} \|\boldsymbol{U} - \overline{\boldsymbol{U}}\|_F^2 \;\; \text{s.t.} \;\; \|\boldsymbol{x}_t - \boldsymbol{U}^\top \boldsymbol{v}_t^{(i)}\|_2^2 \leq \epsilon \; . \tag{3.37}$$

Note the indexing, as $\boldsymbol{U}_t^{(i)}$ is computed after $\boldsymbol{v}_t^{(i)}$, as the latter appears before the former in the loop in Algorithm 3.2. To avoid clutter we will drop the time and iteration indices. We will also use $\overline{\boldsymbol{U}} = \boldsymbol{U}_{t-1}$ to distinguish the time indexing. From Eq. (3.37) it is seen that the problem is convex. To show this, let $\mathcal{I}_t$ denote the index of observed $\boldsymbol{x}_t$ entries, then

$$\begin{aligned}
&\min_{\{\boldsymbol{u}_i\}_{i\in\mathcal{I}_t}} \sum_{i\in\mathcal{I}_t} \|\boldsymbol{u}_i - \overline{\boldsymbol{u}}_i\|_2^2 \;\; \text{s.t.} \;\; \sum_{i\in\mathcal{I}_t} (x_i - \boldsymbol{u}_i^\top \boldsymbol{v})^2 \leq \epsilon \\
&\equiv \min_{\{\boldsymbol{u}_i\}_{i\in\mathcal{I}_t}} \sum_{i\in\mathcal{I}_t} \boldsymbol{u}_i^\top \boldsymbol{I} \boldsymbol{u}_i - 2\boldsymbol{u}_i^\top \overline{\boldsymbol{u}}_i + \overline{\boldsymbol{u}}_i^\top \overline{\boldsymbol{u}}_i \\
&\qquad \text{s.t.} \;\; \sum_{i\in\mathcal{I}_t} \boldsymbol{u}_i^\top \boldsymbol{v}\boldsymbol{v}^\top \boldsymbol{u}_i - 2x_i \boldsymbol{u}_i^\top \boldsymbol{v} + x_i^2
\end{aligned} \tag{3.38}$$

and note that both $\boldsymbol{I}$ and $\boldsymbol{v}_t\boldsymbol{v}_t^\top$ are positive semidefinite. Moreover the feasible set is always nonempty. In fact, for any given $\boldsymbol{v}_t^{(i)}$, the feasible set will have infinitely many elements. To see this, let $\epsilon = 0$ and note that this gives M_t linear equations in dM_t unknowns. Since all of these solutions lie within the feasible set, the result follows. As a consequence, Slater's condition is satisfied and strong duality holds for this problem and we can characterize the solution via Karush Kuhn Tucker (KKT) conditions [16]. For this problem these conditions read as

1. $\nabla_{\boldsymbol{U}^\star} \mathcal{L} = 0$
2. $\|\boldsymbol{x}_t - \boldsymbol{U}^{\star\top} \boldsymbol{v}_t\|_2^2 \leq \epsilon$
3. $\lambda^\star \geq 0$

4. $\lambda^\star \left[\|\boldsymbol{x}_t - \boldsymbol{U}^{\star\top}\boldsymbol{v}_t\|_2^2 - \epsilon\right] = 0$

Using the first condition, which needs to hold for primal feasibility, it is easily shown that the solution satisfies

$$\boldsymbol{U} = [\lambda^{-1}\boldsymbol{I} + \boldsymbol{v}_t\boldsymbol{v}_t^\top]^{-1}(\lambda^{-1}\overline{\boldsymbol{U}} + \boldsymbol{v}_t\boldsymbol{x}_t^\top) \ ,$$

which can be re-written, using Sherman-Morrison identity, as

$$\boldsymbol{U} = \overline{\boldsymbol{U}} + \lambda\boldsymbol{v}_t\boldsymbol{x}_t^\top - \frac{\lambda}{1+\lambda\boldsymbol{v}_t^\top\boldsymbol{v}_t}(\boldsymbol{v}_t\boldsymbol{v}_t^\top)\overline{\boldsymbol{U}} - \frac{\lambda^2}{1+\lambda\boldsymbol{v}_t^\top\boldsymbol{v}_t}(\boldsymbol{v}_t\boldsymbol{v}_t^\top)(\boldsymbol{v}_t\boldsymbol{x}_t^\top) \tag{3.39}$$

This alternative expression is useful, as the inverse term containing λ disappears. Now, the third and fourth KKT conditions imply

$$\|\boldsymbol{x}_t - \boldsymbol{U}^\top\boldsymbol{v}_t\|_2^2 = \epsilon \ . \tag{3.40}$$

This also satisfy condition number two. This means, the solution of the optimization problem has an approximation error that is equal to the maximum tolerance ϵ. As $\boldsymbol{U}$ is always updated after $\boldsymbol{v}_t$ this means the training error of our algorithm at *each* time step will be $\sqrt{\epsilon}$. This also suggests, we can find the value of Lagrange multiplier by plugging Eq. (3.39) into Eq. (3.40). Now define the constants c_1-c_4 as

$$c_1 = \boldsymbol{v}_t^\top\overline{\boldsymbol{U}}\boldsymbol{x}_t \ , \ c_2 = \|\boldsymbol{v}_t\|_2^2 \ , \ c_3 = \|\boldsymbol{x}_t\|_2^2 \ , \ c_4 = \|\overline{\boldsymbol{U}}^\top\boldsymbol{v}_t\|_2^2 \ . \tag{3.41}$$

We need to calculate the equation

$$\boldsymbol{v}_t^\top\boldsymbol{U}\boldsymbol{U}^\top\boldsymbol{v}_t - 2\boldsymbol{x}_t^\top\boldsymbol{U}^\top\boldsymbol{v}_t - (\epsilon - c_3) = 0 \ .$$

The first two terms evaluate as

$$\begin{aligned}
\boldsymbol{v}_t^\top \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{v}_t &= c_4 + \lambda c_1 c_2 - \frac{\lambda}{1+\lambda c_2} c_2 c_4 - \frac{\lambda^2}{1+\lambda c_2} c_1 c_2^2 \\
&+ \lambda c_1 c_2 + \lambda^2 c_2^2 c_3 - \frac{\lambda^2}{1+\lambda c_2} c_1 c_2^2 - \frac{\lambda^3}{1+\lambda c_2} c_2^3 c_3 \\
&- \frac{\lambda}{1+\lambda c_2} c_2 c_4 - \frac{\lambda^2}{1+\lambda c_2} c_1 c_2^2 + \frac{\lambda^2}{(1+\lambda c_2)^2} c_2^2 c_4 \\
&+ \frac{\lambda^3}{(1+\lambda c_2)^2} c_1 c_2^3 - \frac{\lambda^2}{1+\lambda c_2} c_1 c_2^2 - \frac{\lambda^3}{1+\lambda c_2} c_2^3 c_3 \\
&+ \frac{\lambda^3}{(1+\lambda c_2)^2} c_1 c_2^3 + \frac{\lambda^4}{(1+\lambda c_2)^2} c_2^4 c_3 \qquad (3.42) \\
-2\boldsymbol{x}_t^\top \boldsymbol{U}^\top \boldsymbol{v}_t &= -\frac{2c_1}{1+\lambda c_2} - \frac{2\lambda c_2 c_3}{1+\lambda c_2} \qquad (3.43)
\end{aligned}$$

Multiplying with the denominator term $(1+\lambda c_2)^2$ and expanding the terms, the final expression is simply a second order polynomial

$$-\epsilon c_2^2 \lambda^2 - 2\epsilon c_2 \lambda + (c_3 + c_4 - 2c_1 - \epsilon) \ . \qquad (3.44)$$

The roots are then given by the formula

$$-\frac{1}{c_2} \pm \frac{1}{\sqrt{\epsilon} c_2} \sqrt{c_3 + c_4 - 2c_1} \qquad (3.45)$$

Since $c_2 = \|\boldsymbol{v}_t\|_2^2 > 0$ the first term is negative. Then the third KKT condition implies, the second term above must be greater than the first term, and thus the polynomial always has one positive and one negative root. The update for the Lagrange multiplier in Eq. (3.14) now follows; note that in that equation we only used two variables defined in Eq. (3.13) for conciseness.

3.8.2 Fixed tolerance update: $\boldsymbol{v}_t$

Similar to Appendix 3.8.1 we study iteration-i of the E-step in Algorithm 3.2; but this time for a fixed $\boldsymbol{U}_t^{(i)}$ and we want to find

$$\boldsymbol{v}_t^{(i+1)} = \arg\min_{\boldsymbol{v}} \|\boldsymbol{v} - \boldsymbol{v}_{t-1}\|_2^2 \ \text{ s.t. } \ \|\boldsymbol{x}_t - \boldsymbol{U}_t^{(i)} \boldsymbol{v}\|_2^2 \leq \epsilon \ . \qquad (3.46)$$

Unlike the case for $\boldsymbol{U}_t$, it is not clear if the solution set is nonempty. In particular, note that in Appendix 3.8.1, we showed that no matter how $\boldsymbol{v}_t$ is chosen the solution set always has infinitely many elements. For $\boldsymbol{v}_t$ this is not true in general. To see this, consider the case where $\boldsymbol{U}_t^\top \boldsymbol{v} = \boldsymbol{x}$ is an over constrained system of linear equations. The minimum error achievable in this case is given by the least squares solution as $\epsilon_{\text{ls}} = \boldsymbol{x}_t^\top [\boldsymbol{I} - \boldsymbol{U}_t^\top (\boldsymbol{U}_t^\top \boldsymbol{U}_t)^{-1} \boldsymbol{U}_t] \boldsymbol{x}_t$. When $\epsilon_{\text{ls}} > \epsilon$ there are no solutions. However, if $\boldsymbol{U}_t$ is updated before $\boldsymbol{v}$, then it is guaranteed that the there is at least a single solution; because for the alternating optimization in Eq. (3.46), we are given that $\|\boldsymbol{x}_t - \boldsymbol{U}_t^{(i)\top} \boldsymbol{v}_t^{(i-1)}\|_2^2 < \epsilon$. As a result, $\boldsymbol{v}_t^{(i-1)}$ is guaranteed to be in the feasible set. Therefore the solution set is nonempty and since the problem is strongly convex, strong duality holds once again due to Slater's condition. The KKT conditions mirror the previous one:

1. $\nabla_{\boldsymbol{v}^\star} \mathcal{L} = 0$
2. $\|\boldsymbol{x}_t - \boldsymbol{U}_t^\top \boldsymbol{v}^\star\|_2^2 \leq \epsilon$
3. $\lambda^\star \geq 0$
4. $\lambda^\star \left[\|\boldsymbol{x}_t - \boldsymbol{U}_t^\top \boldsymbol{v}^\star\|_2^2 - \epsilon \right] = 0$

Letting $\overline{\boldsymbol{v}} = \boldsymbol{v}_{t-1}$ the first condition yields

$$\boldsymbol{v} = [\lambda^{-1} \boldsymbol{I} + \boldsymbol{U}_t \boldsymbol{U}_t^\top]^{-1} [\lambda^{-1} \overline{\boldsymbol{v}} + \boldsymbol{U}_t \boldsymbol{x}_t] \ . \tag{3.47}$$

This time, we cannot apply Sherman-Morrison identity, as $\boldsymbol{U}_t$ is typically not rank-1. Instead, note that $\boldsymbol{U}_t \boldsymbol{U}_t^\top$ is positive semidefinite and admits an eigendecomposition with nonnegative eigenvalues. Let's denote this by $\boldsymbol{U}_t \boldsymbol{U}_t^\top = \boldsymbol{Q} \boldsymbol{\Psi} \boldsymbol{Q}^\top$, and define the constant vectors $\boldsymbol{c}_1 = \boldsymbol{Q}^\top \boldsymbol{U}_t \boldsymbol{x}_t$ and $\boldsymbol{c}_2 = \boldsymbol{Q}^\top \overline{\boldsymbol{v}}$. Then it is straightforward to verify that

$$\boldsymbol{v} = \boldsymbol{Q} [\rho \boldsymbol{I} + \boldsymbol{\Psi}]^{-1} \boldsymbol{c}_1 + \boldsymbol{Q} \rho [\rho \boldsymbol{I} + \boldsymbol{\Psi}]^{-1} \boldsymbol{c}_2 \tag{3.48}$$

where we defined $\rho = 1/\lambda$; note the choice of notation here as ρ is the regularizer, analogous to the one in FP matrix factorization. Once again, from the fourth KKT condition we want to solve the equation

$$\boldsymbol{v}^\top \boldsymbol{U}_t \boldsymbol{U}_t^\top \boldsymbol{v} - 2\boldsymbol{x}_t^\top \boldsymbol{U}_t^\top \boldsymbol{v} - (\epsilon - \|\boldsymbol{x}_t\|_2^2) = 0$$

where the first two terms are evaluated as

$$\begin{aligned}
\boldsymbol{v}^\top \boldsymbol{U}_t \boldsymbol{U}_t^\top \boldsymbol{v} &= \sum_{i=1}^{d} \frac{\psi_i}{(\rho+\psi_i)^2} c_{1,i}^2 + \sum_{i=1}^{d} \frac{\rho^2 \psi_i}{(\rho+\psi_i)^2} c_{2,i}^2 \\
&\quad + 2\frac{\rho\psi_i}{(\rho+\psi_i)^2} c_{1,i} c_{2,i} \\
-2\boldsymbol{x}_t^\top \boldsymbol{U}_t^\top \boldsymbol{v} &= -2\sum_{i=1}^{d} \frac{1}{\rho+\psi_i} c_{1,i}^2 - 2\sum_{i=1}^{d} \frac{\rho}{\rho+\psi_i} c_{1,i} c_{2,i} \ . \qquad (3.49)
\end{aligned}$$

Substituting these into the equation and multiplying with the denominator term $\prod_{j=1}^{d}(\rho+\psi_i)^2$ we get the following result.

Remark: The FT update for $\boldsymbol{v}$ and the ridge regression update in Eq. (3.18) are the same when λ is selected as the root of the following polynomial yielding smallest $\|\boldsymbol{v}\|_2^2$

$$\begin{aligned}
p(\lambda) &= \sum_{i=1}^{d} (-2\rho - \psi_i) c_{1,i}^2 \prod_{j\neq i} (\rho+\psi_j)^2 + \sum_{i=1}^{d} \rho^2 \psi_i c_{2,i}^2 \prod_{j\neq i} (\rho+\psi_j)^2 \\
&\quad - 2\rho^2 \sum_{i=1}^{d} c_{1,i} c_{2,i} \prod_{j\neq i} (\rho+\psi_j)^2 - (\epsilon - \|\boldsymbol{x}_t\|_2^2) \prod_{j=1}^{d} (\rho+\psi_j)^2 \ . \qquad (3.50)
\end{aligned}$$

Part II

Nonlinear Kalman Filtering

Chapter 4

Nonlinear Kalman Filtering with Divergence Minimization

4.1 Introduction

In Part I, I have considered the problem of dynamic matrix factorization with applications time series analysis. In particular I have introduced the Collaborative Kalman Filter in Chapter 2, which is an instance of the more general Kalman filtering problem [66]. Modeling and analysis of time-varying signals is an important subfield of signal processing and the problem arises in many different forms, such as telecommunication systems, robot motion control, and target tracking. Kalman filter is widely adopted for such problems as it is optimal for a large class of nonstationary state-space models and it can also be used in the online setting. Furthermore, given the growth in sequential data, Kalman filters have also become attractive for machine learning problems, such as natural language processing [12], collaborative filtering [45], and topic modeling [14].

For the model in Chapter 2, the likelihood of observations depend on the latent factors in a nonlinear way; this is but one scenario under the nonlinear Kalman filtering framework. There are a wide range of applications where we deal with nonlinear

systems; the collaborative filtering model contains a bilinear term [70] and in radar tracking distance and bearing measurements require a Cartesian-to-polar transformation [73]. In these cases there is no longer an optimal solution, and approximation schemes are necessary. The nonlinear problem has been studied extensively in the literature, resulting in well-known filtering algorithms such as the Extended Kalman Filter (EKF) [113] and Unscented Kalman Filter (UKF) [65]. On the other hand, Particle Filters (PF) have also been developed [8], which are nonparametric and can represent any probability distribution using a discrete set of points (particles). Also, for filters such as EKF and UKF, the emphasis is on approximating the nonlinear functions, whereas for particle filters it is on approximating the posterior.

While particle filters can approximate arbitrary densities, it may still be important to find the best parametric distribution according to a particular objective function. (For example, the complexity of a particle filter can be too high, or when the system parameters are known with some uncertainty, particle filters can be less robust to errors.) This has been a major goal in Bayesian inference, where the exact posterior distribution is usually intractable and approximated by a known, simpler distribution. Two established ways to handle this problem are Variational Inference [64] and Expectation Propagation [84], in which the Kullback-Leibler (KL) divergence between the true posterior and the approximating distribution is minimized. Ideas from approximate inference have also been used in Kalman filtering [11], [111], [106]. In fact, Chapter 2 is an instance of this, where the approximate posteriors can be obtained in analytical form. However, a thorough analysis of posterior optimization for nonlinear Kalman filters, based on divergence measures as objective function, has not been considered before.

In this chapter we fill this gap by presenting three algorithms for nonlinear Kalman filtering based on three respective divergence measures for posterior approximation, each based on a parametric form (in our case, a multivariate Gaussian). These algorithms are approximation-free in that, they directly optimize the given divergence

measure. We consider the following: (i) the forward KL divergence as used in variational inference, (ii) the reverse KL divergence as used in expectation-propagation, and (iii) the alpha divergence, which is a generalized family that contains the former two as special cases. We also show that well-known algorithms such as the EKF and UKF are actually solving approximations to KL divergence minimization problems which further motivates our study.[1]

The main machinery we use for obtaining these unbiased divergence minimization algorithms is importance sampling. However, the resulting algorithms are all computationally lighter than particle filtering since (i) no resampling is necessary, and (ii) the number of unnecessary samples can be reduced by our proposed adaptive sampling procedure. We show advantages of our algorithms for target tracking and options pricing problems compared with the EKF, UKF and PF.

We organize this chapter as follows: In Section 4.2 we define our filtering framework by reviewing the Kalman filter and discussing its non-linear variants. In particular, we discuss parametric approaches, also called assumed density filters, and nonparametric approaches, also called particle filters. In Section 4.3 we present three divergence minimization filters based on the forward and reverse KL divergences and the alpha divergence. We list the resulting algorithms and their computational cost in Section 4.4. Section 4.5 contains a number of experiments to show how our filters compete with each other and with standard approaches. Finally we conclude in Section 4.6.

[1]Note that even if our proposed algorithms directly optimize divergence measures, the Gaussian posteriors obtained are still inexact.

4.2 Kalman Filtering

4.2.1 Basic Linear Framework

The Kalman filter [66] has been developed and motivated as an optimal filter for linear systems. A key property is that this optimality is assured for general state-space models which are not necessarily stationary. This has made Kalman filtering widely applicable to a range of applications where a linear system model is adequate. In Section 2.2.2 the state space model has been presented in a way that is convenient for the development of CKF. Here we consider the more general form which is given as

$$\begin{aligned} \boldsymbol{x}_t &= \boldsymbol{F}_t \boldsymbol{x}_{t-1} + \boldsymbol{w}_t \\ \boldsymbol{y}_t &= \boldsymbol{H}_t \boldsymbol{x}_t + \boldsymbol{v}_t \end{aligned} \tag{4.1}$$

where $\boldsymbol{w}_t$ and $\boldsymbol{v}_t$ are independent zero-mean Gaussian random vectors with covariances $\boldsymbol{Q}_t$ and $\boldsymbol{R}_t$ respectively. The latent variable $\boldsymbol{x}_t \in \mathbb{R}^d$ is the unobserved state of the system. The vector $\boldsymbol{y}_t \in \mathbb{R}^{d'}$ constitutes the measurements made by the system.

The two main tasks of Kalman filtering are prediction and posterior calculation [113],

$$\begin{aligned} p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) &= \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})\ p(\boldsymbol{x}_{t-1}|\boldsymbol{y}_{1:t-1})\ d\boldsymbol{x}_{t-1}, \\ p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) &= \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)\ p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{\int p(\boldsymbol{y}_t|\boldsymbol{x}_t)\ p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\ d\boldsymbol{x}_t}\ . \end{aligned} \tag{4.2}$$

When the initial distribution $p(\boldsymbol{x}_0)$ is Gaussian these calculations are all in closed form and Gaussian, which is an attractive feature of the linear Kalman filter. The exact equations are

$$\begin{aligned} \text{Predict:} \quad & \boldsymbol{x}_{t|t-1} = \boldsymbol{F}_t \boldsymbol{x}_{t-1|t-1} \\ & \boldsymbol{P}_{t|t-1} = \boldsymbol{F}_t \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t^\top + \boldsymbol{Q}_t \end{aligned} \tag{4.3}$$

$$\begin{aligned} \text{Update:} \quad & \boldsymbol{x}_{t|t} = \boldsymbol{x}_{t|t-1} + \boldsymbol{K}_t(\boldsymbol{y}_t - \boldsymbol{y}_{t|t-1}) \\ & \boldsymbol{P}_{t|t} = \boldsymbol{P}_{t|t-1} - \boldsymbol{K}_t \boldsymbol{S}_t \boldsymbol{K}_t^\top \end{aligned} \tag{4.4}$$

where the auxiliary variables are

$$\begin{aligned} \boldsymbol{y}_{t|t-1} &= \boldsymbol{H}_t \boldsymbol{x}_{t|t-1} \\ \boldsymbol{S}_t &= \boldsymbol{H}_t \boldsymbol{P}_t \boldsymbol{H}_t^\top + \boldsymbol{R}_t \\ \boldsymbol{K}_t &= \boldsymbol{P}_{t|t-1} \boldsymbol{H}_t^\top \boldsymbol{S}_t^{-1} \ . \end{aligned} \tag{4.5}$$

Here, $\boldsymbol{y}_{t|t-1}$ is called the measurement prediction, $\boldsymbol{S}_t$ is the measurement covariance, and $\boldsymbol{K}_t$ is the Kalman gain. The prediction step follows from the linear transformation of a Gaussian random variable, while the update step from Bayes' rule. Further, the update step is obtained by refining the prediction with the innovation term $(\boldsymbol{y}_t - \boldsymbol{y}_{t|t-1})$. Also worthwhile to note that, applying Bayes' rule directly gives the update in Section 2.2.2, and to obtain the version in Eq. (4.4) an application of Matrix Inversion Lemma is necessary.

As we will see in the upcoming sections, when we consider the nonlinear variant of this problem, applying Bayes' rule will no longer give a closed-form update, as the denominator in Eq. (4.2)—also known as the partition function—will be difficult to compute.

4.2.2 Nonlinear Framework

For many problems the measurements $\boldsymbol{y}_t$ involve nonlinear functions of $\boldsymbol{x}_t$. In this case the Kalman filter becomes nonlinear and the closed-form posterior calculation discussed above no longer applies. The nonlinear state-space model is

$$\begin{aligned} \boldsymbol{x}_t &= \boldsymbol{F}_t \boldsymbol{x}_{t-1} + \boldsymbol{w}_t \\ \boldsymbol{y}_t &= \boldsymbol{h}(\boldsymbol{x}_t) + \boldsymbol{v}_t \end{aligned} \tag{4.6}$$

where the noise process is the same as in Eq. (4.1), but $\boldsymbol{h}(\cdot)$ is a nonlinear function of $\boldsymbol{x}_t$.[2] While formally Bayes' rule lets us write

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) = \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t)\ p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})}{\int p(\boldsymbol{y}_t|\boldsymbol{x}_t)\ p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})\ d\boldsymbol{x}_t} \tag{4.7}$$

the normalizing constant computation is now intractable and the distribution $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$ is not known. Although the nonlinearity in $\boldsymbol{h}(\cdot)$ may be required by the problem, a drawback is the loss of fast and exact analytical calculations. In Section 4.3 we introduce three related techniques to approximating $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t})$, but first we review two standard approaches to the problem.

4.2.3 Parametric Approach: Assumed Density Filtering

To address the computational problem posed by Eq. (4.7) Assumed Density Filters (ADF) project the nonlinear update equation to a tractable distribution. Building on the linear Gaussian state-space model, Gaussian assumed density filtering has found wide applicability [81], [113], [65], [61], [46]. The main ingredient here is an assumption of *joint Gaussianity* of the latent and observed variables. This takes the form,

$$p(\boldsymbol{x}_t, \boldsymbol{y}_t) \sim N\left(\begin{bmatrix} \boldsymbol{\mu}_x \\ \boldsymbol{\mu}_y \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{xx} & \boldsymbol{\Sigma}_{xy} \\ \boldsymbol{\Sigma}_{yx} & \boldsymbol{\Sigma}_{yy} \end{bmatrix}\right). \tag{4.8}$$

where we have suppressed some time indices and conditioning terms. Under this joint Gaussian assumption, by standard computations the conditional distribution $p(\boldsymbol{x}_t|\boldsymbol{y}_t)$ is

$$\begin{aligned} p(\boldsymbol{x}_t|\boldsymbol{y}_t) &= N\left(\boldsymbol{\mu}_{x|y}, \boldsymbol{\Sigma}_{x|y}\right) \\ \boldsymbol{\mu}_{x|y} &= \boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}(\boldsymbol{y}_t - \boldsymbol{\mu}_y) \\ \boldsymbol{\Sigma}_{x|y} &= \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}. \end{aligned} \tag{4.9}$$

[2] We focus on measurement nonlinearity in this chapter, assuming the same state space model. The techniques described here can be extended to nonlinearity in the state space as well.

In this case, the conditional distribution is also the posterior distribution of interest. Comparing (4.9) with (4.4) we see that Gaussian ADF corresponds to approximating the three auxiliary variables. We summarize this below:

$$\begin{aligned}
\text{Predict:} \quad \boldsymbol{x}_{t|t-1} &= \boldsymbol{F}_t \boldsymbol{x}_{t-1|t-1} \\
\boldsymbol{P}_{t|t-1} &= \boldsymbol{F}_t \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t^\top + \boldsymbol{Q}_t \qquad (4.10) \\
\text{Update:} \quad \boldsymbol{x}_{t|t} &= \boldsymbol{x}_{t|t-1} + \boldsymbol{K}_t (\boldsymbol{y}_t - \boldsymbol{y}_{t|t-1}) \\
\boldsymbol{P}_{t|t} &= \boldsymbol{P}_{t|t-1} - \boldsymbol{K}_t \boldsymbol{S}_t \boldsymbol{K}_t^\top \qquad (4.11) \\
\text{Auxiliary:} \quad \boldsymbol{y}_{t|t-1} &= \boldsymbol{\mu}_y = \boldsymbol{h}(\boldsymbol{x}_{t|t-1}) \\
\boldsymbol{H}_t &= \boldsymbol{\Sigma}_{yx} \boldsymbol{\Sigma}_{xx}^{-1} \\
\boldsymbol{S}_t &= \boldsymbol{\Sigma}_{yy} = \boldsymbol{\Sigma}_{yx} \boldsymbol{\Sigma}_{xx}^{-1} \boldsymbol{\Sigma}_{xy} + \boldsymbol{R}_t \\
\boldsymbol{K}_t &= \boldsymbol{\Sigma}_{xy} \boldsymbol{\Sigma}_{yy}^{-1} \ . \qquad (4.12)
\end{aligned}$$

Note that the filter structure in Eqs. (4.3)-(4.4) is unchanged, as expected. Moreover, it is straightforward to show that the relation between the auxiliary variables in Eq. (4.5) and Eq. (4.12) is preserved.

There are several methods for making this approximation. We briefly review the two most common here: the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF). The EKF approximates $\boldsymbol{h}$ using the linearization

$$\boldsymbol{h}(\boldsymbol{x}_t) \approx \boldsymbol{h}(\boldsymbol{x}_0) + \boldsymbol{H}(\boldsymbol{x}_0)(\boldsymbol{x}_t - \boldsymbol{x}_0), \qquad (4.13)$$

where $\boldsymbol{H}(\boldsymbol{x}_0)$ is the Jacobian matrix evaluated at the point $\boldsymbol{x}_0$. For example, $\boldsymbol{x}_0$ could be the mean of the prior $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$. Plugging this approximation directly into the likelihood of $\boldsymbol{y}_t$, the form of a linear Kalman filter is recovered and a closed form Gaussian posterior can be calculated.

As discussed in [65], the first-order approximation made by the EKF is often poor and performance can suffer as a result. Instead, they propose to estimate the quantities in (4.8) with the "unscented transform"—a numerical quadrature method. The result is the UKF, which has similar computational cost as the EKF and higher

accuracy. Based on the calculated Gaussian prior $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1}) = N(\boldsymbol{x}_t|\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_{xx})$, the UKF selects a discrete set of sigma points at which to approximate $\boldsymbol{\mu}_y$, $\boldsymbol{\Sigma}_{yy}$ and $\boldsymbol{\Sigma}_{xy}$. Let $d_x = \dim(\boldsymbol{x}_t)$ and $N_s = 2 \times d_x + 1$. These sigma points $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^{N_s}$ are

$$x^s = \begin{cases} \boldsymbol{\mu}_x & \text{for } s = 0 \\ \boldsymbol{\mu}_x + [\sqrt{(d_x + \lambda)\boldsymbol{\Sigma}_{xx}}]_s & \text{for } s = 1, \ldots, d_x \\ \boldsymbol{\mu}_x - [\sqrt{(d_x + \lambda)\boldsymbol{\Sigma}_{xx}}]_{s-d_x} & \text{for } s = d_x + 1, \ldots, 2d_x \end{cases}$$

$$\sum_{s=1}^{N_s} w_m^s \boldsymbol{x}^s = \boldsymbol{\mu}_{xx} \quad , \quad \sum_{i=1}^{N_s} w_c^s (\boldsymbol{x}^s - \boldsymbol{\mu}_{xx})(\boldsymbol{x}^s - \boldsymbol{\mu}_{xx})^\top = \boldsymbol{\Sigma}_{xx} \ . \tag{4.14}$$

The vector $[\sqrt{(d_x + \lambda)\Sigma}]_s$ corresponds to the s-th column of the Cholesky decomposition of the matrix $\sqrt{(d_x + \lambda)\Sigma}$. Positive weights $w_\bullet^s$ are also defined for each $\boldsymbol{x}^s$. The constant λ controls these sigma point locations, as well as the weights (along with additional fixed parameters). These N_s locations are used to empirically approximate all means and covariances in Eq. (4.8). Once y_t is measured, the approximation of $p(\boldsymbol{x}_t|\boldsymbol{y}_t)$ can then be calculated using Eq. (4.9). Next, the quantities in the approximation of (4.8) are given by

$$\begin{aligned} \boldsymbol{\mu}_y &= \sum_{s=1}^{N_s} w_m^s h(\boldsymbol{x}^s) \ , \\ \boldsymbol{\Sigma}_{yy} &= \sum_{s=1}^{N_s} w_c^s (\boldsymbol{h}(\boldsymbol{x}^s) - \boldsymbol{\mu}_y)(\boldsymbol{h}(\boldsymbol{x}^s) - \boldsymbol{\mu}_y)^\top + \boldsymbol{R}_t \ , \\ \boldsymbol{\Sigma}_{xy} &= \sum_{s=1}^{N_s} w_m^s (\boldsymbol{x}^s - \boldsymbol{\mu}_x)(\boldsymbol{h}(\boldsymbol{x}^s) - \boldsymbol{\mu}_y)^\top \ . \end{aligned} \tag{4.15}$$

Finally these values can be used in (4.10)-(4.12) to carry out the filtering.

There are many extensions to the UKF framework such as Cubature Kalman Filtering [63] and Quadrature Kalman Filtering [46], which use different numerical quadratures to carry out the approximation, but still correspond to the joint Gaussian assumption of Eq. (4.8). With that said, however, not all Gaussian ADFs make a joint Gaussianity assumption. For example, methods based on Expectation Propagation use moment matching to obtain a Gaussian posterior approximation without

modifying the joint likelihood distribution [84], [54]. We focus on an a similar method in Section 4.3.2.

4.2.4 Nonparametric Approach: Particle Filtering

We have seen that the main theme of ADF is approximating the posterior with a pre-specified joint probability density; when this joint density is Gaussian then $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) \approx N(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. Since the approximating distribution is pre-specified we call this the parametric approach. On the other hand, the nonparametric approach approximates the posterior using a discrete set of points

$$p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t}) \;\approx\; \sum_{s=1}^{S} w_t^s \;\delta(\boldsymbol{x}_t; \boldsymbol{x}_t^s). \tag{4.16}$$

The positive weights w_t^s sum to one, and $\delta(\boldsymbol{x}_t; \boldsymbol{x}_t^s)$'s are point masses at locations $\boldsymbol{x}_t^s$'s. The weights are calculated at every time using importance sampling. When applied to the Kalman filtering problem the resulting filters are called Particle Filters [43]. Unlike the ADFs, such Monte Carlo based methods guarantee convergence to the true posterior as the number samples N_s grows [8], [4].

Let us first review the importance sampling procedure. Assume that $p(\boldsymbol{x})$ is a PDF which is difficult to sample from. Then we can approximate this distribution using samples from another PDF $\pi(\boldsymbol{x})$—which is called the importance density—and weighing them accordingly:

$$p(\boldsymbol{x}) = \sum_{s=1}^{S} w_s \;\delta(\boldsymbol{x}; \boldsymbol{x}^s) \quad \text{s.t.} \quad \boldsymbol{x}^s \overset{iid}{\sim} \pi(\boldsymbol{x}) \quad \text{and} \quad w_s \propto \frac{p(\boldsymbol{x}^s)}{\pi(\boldsymbol{x}^s)} \;. \tag{4.17}$$

Convergence of (4.17) is a standard result in MCMC literature [4].

It is possible to implement a particle filter solely based on Eq. (4.17), which is called sequential importance sampling (SIS) particle filter. However in practice it suffers from particle degeneracy [8]; i.e. after several iterations we are left with a single particle with nonzero weight. To overcome this, we can resample from the posterior distribution N_s times according to the current weights and assign uniform

weights to the new samples. This version is called sequential importance resampling (SIR) particle filter.

We next describe SIR in detail. At time t we have the posterior $p(\boldsymbol{x}_t|\boldsymbol{y}_t) = \sum_{s=1}^{S} \bar{w}_t^s \delta_{\bar{\boldsymbol{x}}_t^s}$. We can then sample S particles from this discrete distribution, which yields

$$p(\boldsymbol{x}_t|\boldsymbol{y}_t) = \sum_{s=1}^{S} (1/S)\ \delta(\boldsymbol{x}_t; \boldsymbol{x}_t^s) \quad \text{s.t.} \quad \boldsymbol{x}_t^s \overset{iid}{\sim} \sum_{s=1}^{S} \tilde{w}_t^s \delta_{\bar{\boldsymbol{x}}_t^s}\ . \tag{4.18}$$

This is the resampling step where $\bar{w}_t^s$ are replaced by $(1/S)$ in order to prevent weight decay to zero. The prior for the next time is obtained by propagating the particles through the state equation in Eq. (4.6).

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_t) = \sum_{s=1}^{S} (1/S)\ \delta(\boldsymbol{x}_{t+1}; \boldsymbol{F}_t \boldsymbol{x}_t^s + \boldsymbol{q}_t^s) \quad \text{s.t.} \quad \boldsymbol{q}_t^s \overset{iid}{\sim} N(\boldsymbol{0}, \boldsymbol{Q}_t)\ . \tag{4.19}$$

Using the discrete prior distribution in Eq. (4.19) and applying Bayes' rule gives

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{y}_{t+1}) = \sum_{s=1}^{S} \bar{w}_t^s\ \delta(\boldsymbol{x}_{t+1}; \boldsymbol{F}_t \boldsymbol{x}_t^s + \boldsymbol{q}_t^s) \quad \text{s.t.} \quad \bar{w}_t^s \propto p(\boldsymbol{y}_{t+1}|\boldsymbol{F}_t \boldsymbol{x}_t^s + \boldsymbol{q}_t^s)\ . \tag{4.20}$$

This way the estimate of posterior is obtained. A particular benefit of using the prior as proposal distribution in Eq. (4.20) is that, the importance weights are solely determined by the likelihood term $p(\boldsymbol{y}_{t+1}|\boldsymbol{F}_t \boldsymbol{x}_t^\bullet + \boldsymbol{q}_t^\bullet)$. This observation will play an important role when we present the divergence minimization-based filters, upcoming next.

4.3 Three Filters based on Divergence Minimization

In this section we discuss the three proposed divergence minimization approaches to the nonlinear Kalman filtering problem. These include the two directions of the Kullback-Leibler (KL) divergence as well as the related alpha divergence that contains

both KL divergences as limiting cases. In all cases, our goal is to approximate the intractable posterior distribution $p(\boldsymbol{x}_t|\boldsymbol{y}_t)$ with a multivariate Gaussian distribution $q(\boldsymbol{x}_t) = N(\boldsymbol{x}_{t|t}, \boldsymbol{P}_{t|t})$, using these three divergences as potential quality measures. In the following three subsections, we first present one divergence objective and review its tractability issues, followed by our approach to resolving this issue.

4.3.1 Filter 1: Forward KL Divergence Minimization

Given two distributions $p(\boldsymbol{x}|\boldsymbol{y})$ and $q(\boldsymbol{x})$, the forward KL divergence ($KL[q\|p]$ in short) is defined as

$$\mathrm{KL}[q(\boldsymbol{x})\|p(\boldsymbol{x})] = \int q(x) \log \frac{q(\boldsymbol{x})}{p(\boldsymbol{x}|\boldsymbol{y})} d\boldsymbol{x} \ . \tag{4.21}$$

The KL divergence is always nonnegative, becomes smaller the more $q(\boldsymbol{x})$ and $p(\boldsymbol{x})$ overlap, and equals zero if and only if $q(\boldsymbol{x}) = p(\boldsymbol{x})$. These properties of the KL divergence make it a useful tool for measuring how similar two distributions are. It is not a distance measure however, as $\mathrm{KL}[q\|p] \neq \mathrm{KL}[p\|q]$; we discuss the latter in Section 4.3.2. In Bayesian machine learning, minimizing an objective of this form over $q(\boldsymbol{x})$ is know as Variational Inference (VI) [112]. In this case, $p(\boldsymbol{x}|\boldsymbol{y})$ corresponds to an unknown posterior distribution of the model parameters, and $q(\boldsymbol{x})$ is its simpler approximation.

For the nonlinear Kalman filtering problem, the posterior we want to find is $p(\boldsymbol{x}_t|\boldsymbol{y}_t)$; which is intractable due to the denominator term. Therefore, $\mathrm{KL}[q\|p]$ is not calculable either. As we reviewed in detail in Section 2.4, VI instead uses the identity

$$\ln p(\boldsymbol{y}_t) = \mathcal{L}[q(\boldsymbol{x}_t)] + \mathrm{KL}[q(\boldsymbol{x}_t)\|p(\boldsymbol{x}_t|\boldsymbol{y}_t)], \tag{4.22}$$

where

$$\mathcal{L}[q(\boldsymbol{x}_t)] = \int q(\boldsymbol{x}_t) \log \frac{p(\boldsymbol{y}_t, \boldsymbol{x}_t)}{q(\boldsymbol{x}_t)} d\boldsymbol{x}_t. \tag{4.23}$$

The integration can now be carried out as we have access to $p(\boldsymbol{y}_t, \boldsymbol{x}_t)$; which is defined by the state-space model. Since the marginal $\log p(\boldsymbol{y}_t)$ is constant and $\mathrm{KL}[q\|p] \geq 0$, maximizing $\mathcal{L}[q(\boldsymbol{x}_t)]$ is equivalent to minimizing the KL divergence. The term $\mathcal{L}[\cdot]$ is referred to as variational lower bound (VLB).

While nonlinear Kalman filters have a simply defined joint likelihood $p(\boldsymbol{y}_t, \boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$ at time t, a significant problem still arises in calculating the VLB due to the nonlinear function $\boldsymbol{h}(\cdot)$. That is, if we define $q(\boldsymbol{x}_t) = N(\boldsymbol{x}_{t|t}, \boldsymbol{P}_{t|t})$, then for the Gaussian generative process of Eq. (4.6) we optimize $\boldsymbol{x}_{t|t}$ and $\boldsymbol{P}_{t|t}$ over the function

$$\begin{aligned}\mathcal{L}[q(\boldsymbol{x}_t)] &= \mathbb{E}_q[\log p(\boldsymbol{y}_t, \boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})] - \mathbb{E}_q[\log q(\boldsymbol{x}_t)] \\ &= \mathbb{E}_q[\log p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{y}_{1:t-1})] - \mathbb{E}_q[\log p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})] - \mathbb{E}_q[\log q(\boldsymbol{x}_t)] \\ &= \mathbb{E}_q[\log p(\boldsymbol{y}_t|\boldsymbol{x}_t)] - \mathbb{E}_q[\log p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})] - \mathbb{E}_q[\log q(\boldsymbol{x}_t)] \ . \end{aligned} \tag{4.24}$$

From the last line above we get

$$\begin{aligned}\mathcal{L}[q(\boldsymbol{x}_t)] = &-\frac{1}{2}\mathbb{E}_q[(\boldsymbol{y}_t - \boldsymbol{h}(\boldsymbol{x}_t))^\top \boldsymbol{R}_t^{-1}(\boldsymbol{y}_t - \boldsymbol{h}(\boldsymbol{x}_t))] \\ &+ \mathbb{E}_q[\log p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})] - \mathbb{E}_q[\log q(\boldsymbol{x}_t)] + \text{const.}\end{aligned} \tag{4.25}$$

The terms in the second line are tractable, but in the first line the nonlinear term $\boldsymbol{h}(\boldsymbol{x}_t)$ will often result in an integral without closed form solution. In the variational inference literature, common approaches to fixing this issue typically involve making tractable approximations to $\boldsymbol{h}(\boldsymbol{x}_t)$. For example, one such approximation would be to pick a point $\boldsymbol{x}_0$ and make the first-order Taylor approximation $\boldsymbol{h}(\boldsymbol{x}_t) \approx \boldsymbol{h}(\boldsymbol{x}_0) + \boldsymbol{H}(\boldsymbol{x}_0)(\boldsymbol{x}_t - \boldsymbol{x}_0)$. One then replaces $\boldsymbol{h}(\boldsymbol{x}_t)$ in Eq. (4.25) with this approximation and optimizes $\boldsymbol{q}(\boldsymbol{x}_t)$. In fact, in this case the resulting update of $\boldsymbol{q}(\boldsymbol{x}_t)$ is identical to the EKF. This observation implies a correspondence between variational inference and commonly used approximations to nonlinear Kalman filters such as the EKF. We make this formal in the following theorem.

Theorem 1: Let the joint Gaussian ADF correspond to the class of filters which make the joint distribution assumption in Eq. (4.8). Then, all filters in this class optimize an approximate form of the VLB in Eq. (4.25).

We present a complete proof in Appendix 4.7.1. Theorem 1 is general in that, established ADF methods such as UKF and QKF can be viewed as an approximate minimization of the forward KL divergence. This result applies to EKF as well, and we can specialize a bit further.

Corollary 2: The EKF corresponds to optimizing the objective of Eq. (4.25) using a first order Taylor approximation of $\boldsymbol{h}(\cdot)$.

The proof is in Appendix 4.7.2. Consequently, the existing algorithms *modify* $\mathcal{L}[q(\boldsymbol{x}_t)]$ and so the resulting optimization of this approximation is no longer guaranteed to minimize $\mathrm{KL}[q\|p]$. In this section we fix this issue and propose a method to directly optimize the objective function of Eq. (4.25).

Returning to $\mathrm{KL}[q\|p]$ minimization problem, we first write the VLB in a more compact form

$$\mathcal{L}[q(\boldsymbol{x}_t)] = \mathbb{E}_q[f(\boldsymbol{x}_t)] + \mathbb{E}_q[\log p(\boldsymbol{x}_t)] - \mathbb{E}_q[\log q(\boldsymbol{x}_t)] \tag{4.26}$$

where $f(\boldsymbol{x}_t) = -(1/2)(\boldsymbol{y}_t - \boldsymbol{h}(\boldsymbol{x}_t))^\top \boldsymbol{R}_t^{-1}(\boldsymbol{y}_t - \boldsymbol{h}(\boldsymbol{x}_t))$, prior distribution $p(\boldsymbol{x}_t|\boldsymbol{y}_{1:t-1})$ is compactly written as $p(\boldsymbol{x}_t)$ and the expectations are taken over $q(\boldsymbol{x}_t) = N(\boldsymbol{x}_t|\boldsymbol{x}_{t|t}, \boldsymbol{P}_{t|t})$. However, since $\mathbb{E}_q[f(\boldsymbol{x}_t)]$ does not have a closed form solution, $\nabla\mathcal{L}[\cdot]$ cannot be evaluated analytically. Here the prior distribution of $\boldsymbol{x}_t$ is Gaussian; but due to the term $f(\boldsymbol{x}_t)$ the posterior is not. This is an example of non-conjugate variational inference where the prior and posterior have different parametric distributions. This is in contrast to the CKF in Chapter 2 where conjugate variational inference was applied and updates were obtained in closed form. To resolve this, [89] proposed a stochastic search method for sampling unbiased gradients of the VLB, which can be applied to non-conjugate models. This also allows for approximation-free minimization of the forward KL divergence using stochastic gradient descent. We derive this technique for our problem, which will result in smaller values of $\mathrm{KL}[q\|p]$ than the EKF and UKF.

The proposed solution in [89] is to step in the direction of an unbiased stochastic

gradient. To this end, the observation is made that

$$\nabla\mathcal{L}[q(\boldsymbol{x}_t)] = \mathbb{E}_q[f(\boldsymbol{x}_t)\nabla \log q(\boldsymbol{x}_t)] + \nabla\mathbb{E}_q[\log p(\boldsymbol{x}_t)] - \mathbb{E}_q[\log q(\boldsymbol{x}_t)] \tag{4.27}$$

where the identity $\nabla q(\boldsymbol{x}_t) = q(\boldsymbol{x}_t)\nabla \log q(\boldsymbol{x}_t)$ is used. While the second gradient can be calculated analytically with respect to either $\boldsymbol{x}_{t|t}$ or $\boldsymbol{P}_{t|t}$, the first gradient can be calculated using Monte Carlo integration,

$$\mathbb{E}_q[f(\boldsymbol{x}_t)\nabla \log q(\boldsymbol{x}_t)] \approx \frac{1}{S}\sum_{s=1}^{S} f(\boldsymbol{x}_t^s)\nabla \log q(\boldsymbol{x}_t^s) \quad \text{s.t.} \quad \boldsymbol{x}_t^s \overset{iid}{\sim} q(\boldsymbol{x}_t). \tag{4.28}$$

A second observation made by [89] is that the variance of these samples may be too high that S needs to be a large number to make this approximation computationally feasible. For this reason employing variance reduction methods is crucial. One way is to use a control variate $g(\boldsymbol{x}_t)$ that is highly correlated with $f(\boldsymbol{x}_t)$, but has an analytic expectation $\mathbb{E}_q[g(\boldsymbol{x}_t)]$. The gradient of $\mathcal{L}[q(\boldsymbol{x}_t)]$ with a control variate is equal to

$$\begin{aligned}\nabla\mathcal{L}[q(\boldsymbol{x}_t)] = \mathbb{E}_q[(f(\boldsymbol{x}_t) - g(\boldsymbol{x}_t))\nabla \log q(\boldsymbol{x}_t)] \\ + \nabla\mathbb{E}_q[g(\boldsymbol{x}_t)] + \nabla\mathbb{E}_q[\log p(\boldsymbol{x}_t)] - \nabla\mathbb{E}_q[\log q(\boldsymbol{x}_t)]\ .\end{aligned} \tag{4.29}$$

Though this leaves the gradient unchanged, Monte-Carlo sampling of the first term has much smaller variance when $|corr(f, g)|$ is large (calculated using $q(\boldsymbol{x}_t)$). Intuitively, this can be seen by noting that if $f(\boldsymbol{x}_t^s) \approx g(\boldsymbol{x}_t^s)$ at the sampled values $\boldsymbol{x}_t^s$, then $|f(\boldsymbol{x}_t^s) - g(\boldsymbol{x}_t^s)| \ll |f(\boldsymbol{x}_t^s)|$. In this case, the analytic gradient $\lambda\mathbb{E}_q[g(\boldsymbol{x}_t)]$ gives an initial approximation of $\mathbb{E}_q[f(\boldsymbol{x}_t)\nabla \log q(\boldsymbol{x}_t)]$, which is then corrected by the Monte Carlo sampled $\mathbb{E}_q[(f(\boldsymbol{x}_t) - g(\boldsymbol{x}_t))\nabla \log q(\boldsymbol{x}_t)]$. Since $g(\boldsymbol{x}_t)$ is a good approximation of $f(\boldsymbol{x}_t)$ in the region of high probability defined by $q(\boldsymbol{x}_t)$, the analytic approximation captures most information, but is refined by the MC sampled gradient to make the gradient approximation-free.

The requirements on $g(\boldsymbol{x}_t)$ to be a good control variate for $f(\boldsymbol{x}_t)$ are that: (i) it is an approximation of $f(\boldsymbol{x}_t)$, and (ii) the expectation $\mathbb{E}_q[g(\boldsymbol{x}_t)]$ is solvable. There are many possible control variates for the function $f(\boldsymbol{x}_t)$. However, building on the EKF

framework we propose setting

$$g(\boldsymbol{x}_t) = -\frac{1}{2}(\boldsymbol{y}_t - \widetilde{\boldsymbol{h}}(\boldsymbol{x}_t; \boldsymbol{x}_{t|t-1}))^\top \boldsymbol{R}_t^{-1}(\boldsymbol{y}_t - \widetilde{\boldsymbol{h}}(\boldsymbol{x}_t; \boldsymbol{x}_{t|t-1}))$$
$$\widetilde{\boldsymbol{h}}(\boldsymbol{x}_t; \boldsymbol{x}_{t|t-1}) = \boldsymbol{h}(\boldsymbol{x}_{t|t-1}) + \boldsymbol{H}(\boldsymbol{x}_{t|t-1})(\boldsymbol{x}_t - \boldsymbol{x}_{t|t-1}) \ . \tag{4.30}$$

So we use a first order Taylor expansion around the prior mean $\boldsymbol{x}_{t|t-1}$ to approximate $\boldsymbol{h}(\cdot)$. If we define $\widetilde{\boldsymbol{y}}_t = \boldsymbol{y}_t - \boldsymbol{h}(\boldsymbol{x}_{t|t-1}) + \boldsymbol{H}(\boldsymbol{x}_{t|t-1})\boldsymbol{x}_{t|t-1}$, then equivalently we can write

$$g(\boldsymbol{x}_t) = -\frac{1}{2}\left(\widetilde{\boldsymbol{y}}_t - \boldsymbol{H}(\boldsymbol{x}_{t|t-1})\boldsymbol{x}_t\right)^\top \boldsymbol{R}_t^{-1}\left(\widetilde{\boldsymbol{y}}_t - \boldsymbol{H}(\boldsymbol{x}_{t|t-1})\boldsymbol{x}_t\right). \tag{4.31}$$

The expectation is now in closed form. While a better approximation may have greater variance reduction for a fixed number of Monte Carlo samples, we emphasize this is not necessary as the error is compensated by the stochastic term.

Putting everything together, the VLB can be written as follows. In order not to clutter the equations, we will use $\boldsymbol{H} = \boldsymbol{H}(\boldsymbol{x}_{t|t-1})$, $p(\boldsymbol{x}_t) = N(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, and $q(\boldsymbol{x}_t) = (\boldsymbol{\mu}_t', \boldsymbol{\Sigma}_t')$. Then

$$\begin{aligned}
\mathcal{L}[q(\boldsymbol{x}_t)] &= \mathbb{E}_q[f(\boldsymbol{x}_t) - g(\boldsymbol{x}_t)] \\
&\quad - \frac{1}{2}\mathbb{E}_q\left[(\widetilde{\boldsymbol{y}}_t - \boldsymbol{H}\boldsymbol{x}_t)^\top \boldsymbol{R}_t^{-1}(\widetilde{\boldsymbol{y}}_t - \boldsymbol{H}\boldsymbol{x}_t)\right] \\
&\quad - \frac{1}{2}\mathbb{E}_q\left[(\boldsymbol{x}_t - \boldsymbol{\mu}_t)^\top \boldsymbol{\Sigma}_t^{-1}(\boldsymbol{x}_t - \boldsymbol{\mu}_t)\right] - \mathbb{E}_q[q(\boldsymbol{x}_t)] \\
&= \mathbb{E}_q[f(\boldsymbol{x}_t) - g(\boldsymbol{x}_t)] \\
&\quad - \frac{1}{2}\operatorname{tr}\left\{\boldsymbol{H}^\top \boldsymbol{R}_t^{-1}\boldsymbol{H}\boldsymbol{\Sigma}_t'\right\} - \frac{1}{2}\boldsymbol{\mu}_t'\boldsymbol{H}^\top \boldsymbol{R}_t^{-1}\boldsymbol{H}\boldsymbol{\mu}_t' + \widetilde{\boldsymbol{y}}_t^\top \boldsymbol{R}_t^{-1}\boldsymbol{H}\boldsymbol{\mu}_t' \\
&\quad - \frac{1}{2}\operatorname{tr}\left\{\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\Sigma}_t'\right\} - \frac{1}{2}\boldsymbol{\mu}_t'^\top \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t' + \boldsymbol{\mu}_t^\top \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t' + \frac{1}{2}\log|\boldsymbol{\Sigma}_t'|
\end{aligned} \tag{4.32}$$

The gradients of the VLB with respect to the posterior distribution's parameters

$\boldsymbol{\mu}'_t$ and $\boldsymbol{\Sigma}'_t$ are now given by

$$\nabla_{\boldsymbol{\mu}'_t}\mathcal{L}[q(\boldsymbol{x}_t)] = \frac{1}{S}\sum_{s=1}^{S}\big[f(\boldsymbol{x}_t^s) - g(\boldsymbol{x}_t^s)\big]\big[\boldsymbol{\Sigma}_t'^{-1}\boldsymbol{x}^s - \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}'_t\big] + \boldsymbol{H}^\top\boldsymbol{R}_t^{-1}(\widetilde{\boldsymbol{y}}_t - \boldsymbol{H}\boldsymbol{\mu}'_t) + \boldsymbol{\Sigma}_t'^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu}'_t) \tag{4.33}$$

$$\nabla_{\boldsymbol{\Sigma}'_t}\mathcal{L}[q(\boldsymbol{x}_t)] = \frac{1}{2S}\sum_{s=1}^{S}\big[f(\boldsymbol{x}_t^s) - g(\boldsymbol{x}_t^s)\big]\big[\boldsymbol{\Sigma}_t'^{-1}(\boldsymbol{x}^s - \boldsymbol{\mu}'_t)(\boldsymbol{x}_t^s - \boldsymbol{\mu}'_t)^\top\boldsymbol{\Sigma}_t'^{-1} - \boldsymbol{\Sigma}_t'^{-1}\big] + \frac{1}{2}\boldsymbol{\Sigma}_t'^{-1} - \frac{1}{2}\boldsymbol{\Sigma}_t^{-1} - \frac{1}{2}\boldsymbol{H}^\top\boldsymbol{R}_t^{-1}\boldsymbol{H} \tag{4.34}$$

such that $\boldsymbol{x}_t^s \overset{iid}{\sim} q(\boldsymbol{x}_t)$. Once again note that if we omit the first lines in the gradients of Eqs. (4.33) and (4.34) and solve for $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$ by setting them to zero, we get the EKF equations. On the other hand, due to additional terms in Eqs. (4.33) and (4.34) a closed form solution does not exist and we need to use gradient ascent.

Another caveat is, without proper scaling of the gradients we can easily have a numerically unstable algorithm as the covariance matrix can lose its positive definiteness. To fix this we pre-condition the gradients with a symmetric positive definite matrix $\boldsymbol{C}$ and perform the following updates

$$\boldsymbol{\mu}_t'^{(i+1)} = \boldsymbol{\mu}_t'^{(i)} + \rho^{(i)}\ [\boldsymbol{C}^{(i)}\ \nabla_{\boldsymbol{\mu}_t}\mathcal{L}], \tag{4.35}$$

$$\boldsymbol{\Sigma}_t'^{(i+1)} = \boldsymbol{\Sigma}_t'^{(i)} + \rho^{(i)}\ [\boldsymbol{C}^{(i)}\ \nabla_{\boldsymbol{\Sigma}_t}\mathcal{L}\ \boldsymbol{C}^{(i)}]\ . \tag{4.36}$$

We highlight the difference between indices t and i. The first is the time index, while the second is the iteration number at time t since we are using gradient descent. For the conditioning matrix we choose $\boldsymbol{C}^{(i)} = [\boldsymbol{\Sigma}_t'^{(i)}]^{-1}$, which approximates the natural gradient [1] for $\boldsymbol{\mu}'_t$ and $\boldsymbol{\Sigma}'_t$.[3] When the step size satisfies the Robbins-Monro conditions, $\sum_{i=1}^{\infty}\rho^{(i)} = \infty$ and $\sum_{i=1}^{\infty}[\rho^{(i)}]^2 < \infty$, the gradients in Eqs. (4.35) and (4.36) converge to a stationary point of the exact variational lower bound. In practice we can, for example, choose $\rho^{(i)} = (w+i)^{-\eta}$ with $\eta \in (0.5, 1]$ and $w \geq 0$. In simulations we observed that when the natural gradients are employed, a generic

[3]The exact natural gradient for $\boldsymbol{\Sigma}'_t$ is slightly different, as discussed in [115].

schedule for step sizes can be used and no further hand-tuning is necessary. The iterations are run I times, which gives the posterior distribution $q(\boldsymbol{x}_t) = N(\boldsymbol{x}_{t|t-1}, \boldsymbol{P}_{t|t-1})$ with $\boldsymbol{x}_{t|t-1} = \boldsymbol{\mu}_t'^{(I)}$ and $\boldsymbol{P}_{t|t-1} = \boldsymbol{\Sigma}_t'^{(I)}$. We refer to this algorithm as Stochastic Search Kalman Filter (SKF) and summarize it in Algorithm 4.1 .

4.3.2 Filter 2: Reverse KL Divergence Minimization

As mentioned in Section 4.3.1, the KL divergence is not a distance measure because it is not symmetric. The complement of the forward KL divergence defined in (4.21) is the reverse KL divergence:

$$\mathrm{KL}[p(\boldsymbol{x})\|q(\boldsymbol{x})] = \int p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\boldsymbol{x} \ . \tag{4.37}$$

We can see that (4.37) offers an alternative measure of how similar two probability distributions are; therefore we can use it to approximate an intractable posterior distribution. Note that for either objective function in Eqs. (4.21) or (4.37), the optimal solution will be $q(\boldsymbol{x}) = p(\boldsymbol{x}|\boldsymbol{y})$. However, since typically the approximating distribution family does not encompass the exact posterior distribution, the two optimization problems will give different solutions in practice.

In fact the reverse KL divergence has shown to be a better fit for unimodal approximations, while forward KL works better in the multimodal case [13]. Consequently, we can expect that optimizing the reverse KL divergence will be a better choice for the nonlinear Kalman filtering problem (this is supported by our experiments). In Section 4.3.2, finding a stationary point of the forward KL required an iterative scheme for maximizing the variational objective function. The stationary point of the reverse KL has a more interpretable form, as we will show.

To this end, we first note that an exponential family distribution has the form

$$q(\boldsymbol{x}) = h(\boldsymbol{x}) \exp\{\boldsymbol{\eta}^\top s(\boldsymbol{x}) - \log A(\boldsymbol{\eta})\}$$

where $\boldsymbol{\eta}$ is the natural parameter and $s(\boldsymbol{x})$ is the sufficient statistic. Therefore inference in exponential families correspond to determining $\boldsymbol{\eta}$. Substituting this

parametrized form in Eq. (4.37) and taking the derivative with respect to the natural parameter one can show that

$$\begin{aligned}
&\nabla_{\boldsymbol{\eta}}\mathrm{KL}[p(\boldsymbol{x})\|q(\boldsymbol{x})] \\
&\quad= \nabla_{\boldsymbol{\eta}} \int \left[p(\boldsymbol{x})\log p(\boldsymbol{x}) - p(\boldsymbol{x})\log q(\boldsymbol{x})\right] d\boldsymbol{x} \\
&\quad= \nabla_{\boldsymbol{\eta}} \int \left[p(\boldsymbol{x})\log p(\boldsymbol{x}) - p(\boldsymbol{x})\log h(\boldsymbol{x}) - p(\boldsymbol{x})\boldsymbol{\eta}^\top s(\boldsymbol{x}) + p(\boldsymbol{x})\log A(\boldsymbol{\eta})\right] d\boldsymbol{x} \\
&\quad= \int p(\boldsymbol{x})\nabla_{\boldsymbol{\eta}}\log A(\boldsymbol{\eta}) d\boldsymbol{x} - \int p(\boldsymbol{x}) s(\boldsymbol{x}) d\boldsymbol{x} \\
&\quad= \int q(\boldsymbol{x}) s(\boldsymbol{x}) d\boldsymbol{x} \int p(\boldsymbol{x}) d\boldsymbol{x} - \int p(\boldsymbol{x}) s(\boldsymbol{x}) d\boldsymbol{x}
\end{aligned} \tag{4.38}$$

where the exponential family identity for the derivative of the partition function is applied. Applying the stationarity condition $\nabla_{\boldsymbol{\eta}}\mathrm{KL}[p(\boldsymbol{x})\|q(\boldsymbol{x})] = \mathbf{0}$ yields the following

$$\mathbb{E}_q[s(\boldsymbol{x})] = \mathbb{E}_p[s(\boldsymbol{x})] \ . \tag{4.39}$$

This is the well-known moment matching condition which is frequently used in statistics and machine learning [13]—one prominent application being the Expectation Propagation (EP) [84], [53]. A common choice for the approximating exponential family distribution is again Gaussian because it is the maximum entropy distribution for the given first and second order moments [112]. Since a Gaussian is completely specified by its mean and covariance, when the approximating distribution $q(\boldsymbol{x})$ is selected to be Gaussian, the optimal solution is simply found by matching its mean and covariance to that of $p(\boldsymbol{x}|\boldsymbol{y})$.

For the nonlinear Kalman filtering problem, when a Gaussian is used as the approximating posterior, we need to match the moments of $q(\boldsymbol{x}_t)$ (which are $\boldsymbol{x}_{t|t}$ and $\boldsymbol{P}_{t|t}$) to that of the true posterior $p(\boldsymbol{x}_t|\boldsymbol{y}_t)$. However, there is still a difficulty as the true posterior is unknown. Fortunately, MC methods prove useful here as well. Let $\mathbb{E}_{p(\boldsymbol{x}_t|\boldsymbol{y}_t)}[f(\boldsymbol{x}_t)]$ be the expectation we wish to compute with respect to the true posterior. For example, choosing $f(\boldsymbol{x}_t) = \boldsymbol{x}$ and $f(\boldsymbol{x}_t) = \boldsymbol{x}_t\boldsymbol{x}_t^\top - \mathbb{E}[\boldsymbol{x}_t]\mathbb{E}[\boldsymbol{x}_t]^\top$ gives the

mean and covariance respectively. This expectation can be approximated as

$$\begin{aligned}\mathbb{E}_p[f(\boldsymbol{x}_t)] &= \int f(\boldsymbol{x}_t)\frac{p(\boldsymbol{x}_t|\boldsymbol{y}_t)}{\pi(\boldsymbol{x}_t)}\pi(\boldsymbol{x}_t)d\boldsymbol{x}_t, \\ &= \int f(\boldsymbol{x}_t)\frac{[p(\boldsymbol{y}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t)]/\pi(\boldsymbol{x}_t)}{\int p(\boldsymbol{y}_t|\boldsymbol{x}'_t)p(\boldsymbol{x}'_t)d\boldsymbol{x}'_t}\pi(\boldsymbol{x}_t)d\boldsymbol{x}_t, \\ &\approx \sum_{s=1}^{S} f(\boldsymbol{x}_t^s)\frac{[p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)p(\boldsymbol{x}_t^s)]/\pi(\boldsymbol{x}_t^s)}{\sum_{j=1}^{S}[p(\boldsymbol{y}_t|\boldsymbol{x}_t^j)p(\boldsymbol{x}_t^j)]/\pi(\boldsymbol{x}_t^j)} \quad \text{s.t.} \quad \boldsymbol{x}_t^s \overset{iid}{\sim} \pi(\boldsymbol{x}_t) \end{aligned} \tag{4.40}$$

where we define the normalized weights to be

$$w_t^s = \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)p(\boldsymbol{x}_t^s)}{\pi(\boldsymbol{x}_t^s)} \quad , \quad W_t = \sum_{s=1}^{S}\frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)p(\boldsymbol{x}_t^s)}{\pi(\boldsymbol{x}_t^s)} \ .$$

As a result we have an importance sampling procedure. We have reviewed this methodology within the particle filtering context in Section 4.2.4 . In particular, if we choose the proposal distribution to be the prior $\pi(\boldsymbol{x}_t) = p(\boldsymbol{x}_t)$ we get $w^s \propto p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)$ so that the weights are solely determined by the likelihood of the observation. As we can see from Eq. (4.40) the importance sampling is biased as it is a ratio of two approximations, yet it converges to the true expectation $\mathbb{E}_q[f(\boldsymbol{x})]$ almost surely which yields an asymptotically unbiased divergence minimization procedure. We call this the Moment Matching Kalman Filter (MKF) and summarize it in Algorithm 4.2. We observe that a major difference between the MKF and SKF of Section 4.3.1 is that, the former can obtain posterior approximations in a single step similar to the EKF and UKF. This makes MKF considerably faster than SKF. Further, the MKF requires half the computation of particle filtering as it does not require resampling.

4.3.3 Filter 3: Alpha Divergence Minimization

In Sections 4.3.1 and 4.3.2 we showed how nonlinear Kalman filtering can be performed by minimizing the forward and reverse KL divergences. A further generalization is possible by considering the alpha divergence, which contains both KL divergences as special case. Following [53] we define the alpha divergence to be

$$D_\alpha[p(\boldsymbol{x})\|q(\boldsymbol{x})] = \frac{1}{\alpha(1-\alpha)}\left(1 - \int p(\boldsymbol{x})^\alpha q(\boldsymbol{x})^{1-\alpha}d\boldsymbol{x}\right), \tag{4.41}$$

where the parameter α can take any value in $(-\infty, \infty)$. Some special cases are

$$\lim_{\alpha\to 0} D_\alpha[p\|q] = \mathrm{KL}[q\|p] \ , \ \lim_{\alpha\to 1} D_\alpha[p\|q] = \mathrm{KL}[p\|q] \ ,$$
$$D_{\frac{1}{2}}[p\|q] = 2\int (\sqrt{p(\boldsymbol{x})} - \sqrt{q(\boldsymbol{x})})^2 dx = 4\mathrm{Hel}^2[p\|q] \ , \tag{4.42}$$

where $\mathrm{Hel}[p\|q]$ is the Hellinger distance. We see that when $\alpha = 1/2$ we get a valid distance metric. Similar to before, we seek a q-distribution which approximates $p(\boldsymbol{x}|\boldsymbol{y})$, where approximation quality is now measured by the alpha divergence.

We again begin assuming that the approximating distribution is in the exponential family, $q(\boldsymbol{x}) = h(\boldsymbol{x}) \exp\{\boldsymbol{\eta}^\top s(\boldsymbol{x}) - \log A(\boldsymbol{\eta})\}$. Substituting this and taking derivative with respect to the natural parameter yields

$$\begin{aligned}
-\alpha(1-\alpha)\nabla_{\boldsymbol{\eta}} D_\alpha[p(\boldsymbol{x})\|q(\boldsymbol{x})] = \\
&= \nabla_{\boldsymbol{\eta}} \left(\int p(\boldsymbol{x})^\alpha q(\boldsymbol{x})^{1-\alpha} d\boldsymbol{x} - 1 \right) \\
&= \nabla_{\boldsymbol{\eta}} \int p(\boldsymbol{x})^\alpha h(\boldsymbol{x})^{1-\alpha} e^{(1-\alpha)(\boldsymbol{\eta}^\top s(\boldsymbol{x}) - \log A(\boldsymbol{\eta}))} \\
&= \int p(\boldsymbol{x})^\alpha h(\boldsymbol{x})^{1-\alpha} e^{(1-\alpha)(\boldsymbol{\eta}^\top s(\boldsymbol{x}) - \log A(\boldsymbol{\eta}))} (1-\alpha)(s(\boldsymbol{x}) - \mathbb{E}_q[s(\boldsymbol{x})]) \\
&= \int p(\boldsymbol{x})^\alpha q(\boldsymbol{x})^{1-\alpha} (1-\alpha)(s(\boldsymbol{x}) - \mathbb{E}_q[s(\boldsymbol{x})]) \\
&= (1-\alpha) Z_{\widetilde{p}} \int \widetilde{p}(\boldsymbol{x}) \left[s(\boldsymbol{x}) - \mathbb{E}_q[s(\boldsymbol{x})] \right] \\
&= \mathbb{E}_{\widetilde{p}}[s(\boldsymbol{x})] - \mathbb{E}_q[s(\boldsymbol{x})]
\end{aligned} \tag{4.43}$$

where we have defined a new probability distribution $\widetilde{p}(\boldsymbol{x}) = p(\boldsymbol{x})^\alpha q(\boldsymbol{x})^{1-\alpha} / Z_{\widetilde{p}}$. The stationary condition $\nabla_{\boldsymbol{\eta}} D_\alpha[p(\boldsymbol{x})\|q(\boldsymbol{x})] = \mathbf{0}$ now leads to a *generalized* moment matching condition

$$\mathbb{E}_q[s(\boldsymbol{x})] = \mathbb{E}_{\widetilde{p}}[s(\boldsymbol{x})] \ . \tag{4.44}$$

The moment matching condition of Eq. (4.39) was used within the EP algorithm; in this context the new condition of Eq. (4.44) leads to a generalization called Power EP [85]. More recently, [53] used a similar black-box optimization, where they showed

that by varying the value of α, the algorithm varies between variational inference and expectation propagation.

It turns out that, for many practical problems, using a fractional value of α can give better performance than the limiting cases $\alpha = 0$ or $\alpha = 1$. We next describe how we can use α-divergence minimization for nonlinear Kalman filtering. The main problem with the moment matching in Eq. (4.44) is that the q-distribution appears on both sides of the equation, so the solutions cannot be obtained in a single step. Unfortunately an iterative scheme is not guaranteed to converge—also observed by [85]. With that said, if a solution satisfying Eq. (4.44) were to be found, it would be a stationary point of the alpha divergence.

However, we can still proceed with our importance sampling method as follows. Using similar notation, we can write

$$\begin{aligned}
\mathbb{E}_{\widetilde{p}}[f(\boldsymbol{x}_t)] &= \int f(\boldsymbol{x}_t) \frac{p(\boldsymbol{x}_t|\boldsymbol{y}_t)^\alpha q(\boldsymbol{x}_t)^{1-\alpha}}{\pi(\boldsymbol{x}_t)} \pi(\boldsymbol{x}_t) d\boldsymbol{x}_t \\
&= \int f(\boldsymbol{x}_t) \frac{[p(\boldsymbol{y}_t|\boldsymbol{x}_t)^\alpha p(\boldsymbol{x}_t)^\alpha q(\boldsymbol{x}_t)^{1-\alpha}]/\pi(\boldsymbol{x}_t)}{\int p(\boldsymbol{y}_t|\boldsymbol{x}_t)^\alpha p(\boldsymbol{x}_t)^\alpha q(\boldsymbol{x}_t)^{1-\alpha} d\boldsymbol{x}_t} \pi(\boldsymbol{x}_t) d\boldsymbol{x}_t \\
&\approx \sum_{s=1}^{S} f(\boldsymbol{x}_t^s) \frac{[p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)p(\boldsymbol{x}_t^s)]^\alpha q(\boldsymbol{x}_t^s)^{1-\alpha}/\pi(\boldsymbol{x}_t^s)}{\sum_{j=1}^{S}[p(\boldsymbol{y}_t|\boldsymbol{x}_t^j)^\alpha p(\boldsymbol{x}_t^j)^\alpha q(\boldsymbol{x}_t^j)^{1-\alpha}]/\pi(\boldsymbol{x}_t^j)} \quad \text{s.t.} \quad \boldsymbol{x}_t^s \overset{iid}{\sim} \pi(\boldsymbol{x}_t)
\end{aligned} \tag{4.45}$$

where we define the normalized weights to be

$$w_t^s = \frac{[p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)p(\boldsymbol{x}_t^s)]^\alpha q(\boldsymbol{x}_t)^{1-\alpha}}{\pi(\boldsymbol{x}_t^s)} \quad , \quad W_t = \sum_{s=1}^{S} \frac{[p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)p(\boldsymbol{x}_t^s)]^\alpha q(\boldsymbol{x}_t)^{1-\alpha}}{\pi(\boldsymbol{x}_t^s)} \, .$$

The procedure of Eq. (4.40) is a special case of Eq. (4.45) with $\alpha = 1$. However, in the former the $q(\boldsymbol{x})$ does not appear in the weights so that the solution can be obtained in a single iteration. This is not possible in the latter. The latter case similar to the EP and Power EP algorithms, where multiple iterations can be run to update $q(\boldsymbol{x})$. However for our purposes we will still use a single iteration and to match the moments; this way the algorithm will have the same cost as MKF. Furthermore, it also hedges computations against numerical instability. We call this algorithm Alpha

Divergence Kalman Filter (αKF) and summarize it in Algorithm 4.4. We note that the only difference between αKF and MKF is in Step 4.

We can get a better understanding of the new importance sampling by analyzing a special case of the weight coefficients. In particular, assume that we choose the proposal distribution to be the prior, $\pi(\boldsymbol{x}_t) = p(\boldsymbol{x}_t)$ and we also set the initial value for the posterior as $q(\boldsymbol{x}_t) = p(\boldsymbol{x}_t)$. Then the αKF weights are $w_t^s \propto p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)^\alpha$ so once again they are solely determined by the likelihood term. Compare this to MKF weights which are $w_t^s \propto p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)$. We see that all the weights are scaled by the α term. In the limiting case $\alpha \to 1$ this behaves like MKF, and as $\alpha \to 0$ all weights are equal. So the values of α in between acts like a *dampening factor*, which makes the estimates robust when the likelihood term is highly noisy. We will show specific examples of this in the experiments.

4.3.4 Adaptive Sampling

The main parameter in the implementation of sampling based filters —including particle filters and the three filters proposed here— is the number of samples/particles S. Hence it is desirable to have a method for estimating the number of samples necessary for a given degree of accuracy. This computed sample size can be used to adaptively reduce the computation as much as possible, while still maintaining a desired accuracy. As we will see in the experiments, remarkable cost cutting is possible this way. In Figure 4.1 we illustrate the adaptive sampling (AdaSamp) approach for tracking a moving target. At time t, the target makes an abrupt maneuver that requires more particles for accurate tracking; but the number can be reduced afterwards.

Since the approximate Gaussian posterior distribution has a parametric form, we are able to use an adaptive sampling method for the MKF and αKF.[4] To determine the appropriate number of samples, we measure the uncertainty of our mean approxi-

[4]For the SKF the per-iteration sample size is much smaller, so there is less benefit in using this technique.

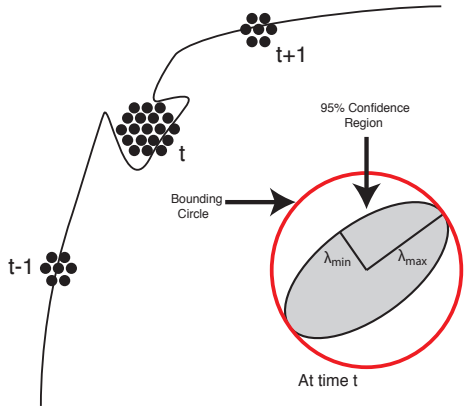


Figure 4.1: Illustration of adaptive sampling. Due to unexpected changes in the target trajectory, more samples may be needed at a given time point. Also shown is the bounding circle for a confidence ellipse.

mation for $q(\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{\mu}'_t, \boldsymbol{\Sigma}'_t)$, where $\boldsymbol{\mu}'_t = \sum_{s=1}^{S} w_t^s \boldsymbol{x}_t^s / W$. For importance sampling, the variance of this estimator is approximately

$$\mathbb{V}(\boldsymbol{\mu}'_t) \;\approx\; \sum_{s=1}^{S} \left[\frac{w_t^s}{W_t}\right]^2 (\boldsymbol{x}_t^s - \boldsymbol{\mu}'_t)(\boldsymbol{x}_t^s - \boldsymbol{\mu}'_t)^\top. \tag{4.46}$$

Here, if S is large enough, the estimator can be approximated as normal by the central limit theorem [4]. We use this fact to compute the radius of a 95% confidence region. Assuming that the estimator is zero mean, which is justified by the asymptotic unbiasedness of the unnormalized importance sampling procedure, we denote the estimator by $\widehat{\boldsymbol{X}} \sim N(\boldsymbol{0}, \mathbb{V}(\boldsymbol{\mu}'_t))$. It follows that

$$P(\widehat{\boldsymbol{X}}^\top \mathbb{V}(\boldsymbol{\mu}'_t)^{-1} \widehat{\boldsymbol{X}} \leq \chi_d^2(p)) = p, \tag{4.47}$$

where $\chi_d^2(p)$ is the quantile function of a chi-squared distribution with d degrees of freedom (set to the state-space dimension), and p is the probability value (for 95% confidence intervals this is set to $p = 0.95$). The region described by Eq. (4.47) is a hyper-ellipsoid, so the maximum possible radius will correspond to

$$r_{\max} = \sqrt{\lambda_{\max}(\mathbb{V}(\boldsymbol{\mu}'_t)) \times \chi_d^2(0.95)}\ . \tag{4.48}$$

Note that this is a conservative estimate, as the hypersphere with radius $r_{\max}$ can be much larger than the hyper-ellipsoid. As an illustration, the bounding circle of a bivariate normal distribution is given in Figure 4.1.

Now assume that, based on a small sample set S_{base}, we wish to estimate the minimum number of samples $S_{\min}$ required to achieve a certain $r_{\max}$. Since the eigenvalues of the covariance matrix decay as $O(1/S)$, we get the relation

$$\frac{r_1}{r_2} \propto \sqrt{\frac{S_2}{S_1}}, \qquad S_{\min} = S_{\text{base}} \times \left[\frac{r_{\text{base}}}{r_{\max}}\right]^2. \tag{4.49}$$

As expected, the smaller radius we desire, the larger sample size we need. Another important point is that the $\mathbb{V}(\boldsymbol{\mu}_t')$ is our confidence in estimating the mean of the posterior, and not the ground truth. The accuracy of estimating the latter is dictated by the measurement noise, and cannot be made arbitrarily small by increasing the sample size. The versions of MKF and αKF using AdaSamp are given in Algorithms 4.3 and 4.5 respectively.

4.4 Algorithms and Complexity

4.4.1 Stochastic Search Kalman Filter

Algorithm 4.1 Stochastic Search Kalman Filter (SKF)

Input: $t = 1, \ldots, T:$ $\boldsymbol{y}_t$ (observations)

$t = 1, \ldots, T:$ $\boldsymbol{F}_t$, $\boldsymbol{h}_t(\cdot)$, $\boldsymbol{Q}_t$, $\boldsymbol{R}_t$ (state-space parameters)

S (samples/inner loop), I (iterations), $(\boldsymbol{\mu}_{0|0}, \boldsymbol{P}_{0|0})$ (initial guess)

$i = 1, \ldots, I:$ $\rho^{(i)}$ (step size)

Output: $t = 1, \ldots, T:$ $p(\boldsymbol{x}_t|\boldsymbol{y}_t) = N(\boldsymbol{x}_{t|t}, \boldsymbol{P}_{t|t})$

Initialize: $p(\boldsymbol{x}_0) = N(\boldsymbol{x}_{0|0}, \boldsymbol{P}_{0|0})$

for $t = 1, \ldots, T$ **do**

  $\boldsymbol{x}_{t|t-1} \leftarrow \boldsymbol{F}_t^\top \boldsymbol{x}_{t-1|t-1}$, $\boldsymbol{P}_{t|t-1} \leftarrow \boldsymbol{F}_t^\top \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t + \boldsymbol{Q}_t$

  $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \leftarrow N(\boldsymbol{x}_{t|t-1}, \boldsymbol{P}_{t|t-1})$

  $\boldsymbol{\mu}_t^{(0)} \leftarrow \boldsymbol{x}_{t|t-1}$, $\boldsymbol{\Sigma}_t^{(0)} \leftarrow \boldsymbol{P}_{t|t-1}$

  $q^{(0)}(\boldsymbol{x}_t) \leftarrow N(\boldsymbol{\mu}_t^{(0)}, \boldsymbol{\Sigma}_t^{(0)})$

  for $i = 1, \ldots, I$ **do**

    $s = 1, \ldots, S$: $\boldsymbol{x}_t^s \overset{iid}{\sim} q^{(i-1)}(\boldsymbol{x}_t)$.

    Compute $\nabla_{\boldsymbol{\mu}_t}^{(i)} \mathcal{L}[q(\boldsymbol{x}_t)]$ using Eq. (4.33).

    Compute $\nabla_{\boldsymbol{\Sigma}_t}^{(i)} \mathcal{L}[q(\boldsymbol{x}_t)]$ using Eq. (4.34).

    $\boldsymbol{C}^{(i)} \leftarrow \boldsymbol{\Sigma}_t^{(i-1)}$

    $\boldsymbol{\mu}_t^{(i)} \leftarrow \boldsymbol{\mu}_t^{(i-1)} + \rho^{(i)} \left[\boldsymbol{C}^{(i)} \nabla_{\boldsymbol{\mu}_t}^{(i)} \mathcal{L}[q(\boldsymbol{x}_t)] \right]$

    $\boldsymbol{\Sigma}_t^{(i)} \leftarrow \boldsymbol{\Sigma}_t^{(i-1)} + \rho^{(i)} \left[\boldsymbol{C}^{(i)} \nabla_{\boldsymbol{\Sigma}_t}^{(i)} \mathcal{L}[q(\boldsymbol{x}_t)] \boldsymbol{C}^{(i)} \right]$

    $q^{(i)}(\boldsymbol{x}_t) \leftarrow N(\boldsymbol{\mu}_t^{(i)}, \boldsymbol{\Sigma}_t^{(i)})$

  end for

  $\boldsymbol{x}_{t|t} = \boldsymbol{\mu}_t^{(I)}$, $\boldsymbol{P}_{t|t} = \boldsymbol{\Sigma}_t^{(I)}$

  $p(\boldsymbol{x}_t|\boldsymbol{y}_t) \leftarrow N(\boldsymbol{x}_{t|t}, \boldsymbol{\Sigma}_{t|t})$

end for

4.4.2 Moment Matching Kalman Filter

Algorithm 4.2 Moment Matching Kalman Filter (MKF)

Input: $t = 1, \ldots, T:$ $\boldsymbol{y}_t$ (observations)

$t = 1, \ldots, T:$ $\boldsymbol{F}_t$, $\boldsymbol{h}_t(\cdot)$, $\boldsymbol{Q}_t$, $\boldsymbol{R}_t$ (state-space parameters)

S (samples), $(\boldsymbol{\mu}_{0|0}, \boldsymbol{P}_{0|0})$ (initial guess)

Output: $t = 1, \ldots, T:$ $p(\boldsymbol{x}_t|\boldsymbol{y}_t) = N(\boldsymbol{x}_{t|t}, \boldsymbol{P}_{t|t})$

Initialize: $p(\boldsymbol{x}_0) = N(\boldsymbol{x}_{0|0}, \boldsymbol{P}_{0|0})$

for $t = 1, \ldots, T$ **do**

$\boldsymbol{x}_{t|t-1} \leftarrow \boldsymbol{F}_t^\top \boldsymbol{x}_{t-1|t-1}$, $\boldsymbol{P}_{t|t-1} \leftarrow \boldsymbol{F}_t^\top \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t + \boldsymbol{Q}_t$

$p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \leftarrow N(\boldsymbol{x}_{t|t-1}, \boldsymbol{P}_{t|t-1})$

$s = 1, \ldots, S$: $\boldsymbol{x}_t^s \overset{iid}{\sim} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$.

$s = 1, \ldots, S$: $w_t^s \leftarrow p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)$, $W_t \leftarrow \sum_{s=1}^S w_t^s$

$s = 1, \ldots, S$: $w_t^s \leftarrow w_t^s / W_t$

$\boldsymbol{x}_{t|t} \leftarrow \sum_{s=1}^S w_t^s \boldsymbol{x}_t^s$

$\boldsymbol{P}_{t|t} \leftarrow \sum_{s=1}^S w_t^s (\boldsymbol{x}_t^s - \boldsymbol{x}_{t|t})(\boldsymbol{x}_t^s - \boldsymbol{x}_{t|t})^\top$

$p(\boldsymbol{x}_t|\boldsymbol{y}_t) \leftarrow N(\boldsymbol{x}_{t|t}, \boldsymbol{\Sigma}_{t|t})$

end for

4.4.3 Moment Matching Kalman Filter with AdaSamp

Algorithm 4.3 Moment Matching Kalman Filter with AdaSamp (MKF)

Input: $t = 1, \ldots, T:$ $\boldsymbol{y}_t$ (observations)

$t = 1, \ldots, T:$ $\boldsymbol{F}_t$, $\boldsymbol{h}_t(\cdot)$, $\boldsymbol{Q}_t$, $\boldsymbol{R}_t$ (state-space parameters)

$(\boldsymbol{\mu}_{0|0}, \boldsymbol{P}_{0|0})$ (initial guess)

S_{base} (initial sample size), $r_{\max}$ (error tolerance)

Output: $t = 1, \ldots, T:$ $p(\boldsymbol{x}_t|\boldsymbol{y}_t) = N(\boldsymbol{x}_{t|t}, \boldsymbol{P}_{t|t})$

Initialize: $p(\boldsymbol{x}_0) = N(\boldsymbol{x}_{0|0}, \boldsymbol{P}_{0|0})$

for $t = 1, \ldots, T$ **do**

$\boldsymbol{x}_{t|t-1} \leftarrow \boldsymbol{F}_t^\top \boldsymbol{x}_{t-1|t-1}$, $\boldsymbol{P}_{t|t-1} \leftarrow \boldsymbol{F}_t^\top \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t + \boldsymbol{Q}_t$

$p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \leftarrow N(\boldsymbol{x}_{t|t-1}, \boldsymbol{P}_{t|t-1})$

`// AdaSamp`

$s = 1, \ldots, S_{\text{base}}$: $\boldsymbol{x}_t^s \overset{iid}{\sim} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$.

$s = 1, \ldots, S_{\text{base}}$: $w_t^s \leftarrow p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)$, $W_t \leftarrow \sum_{s=1}^{S} w_t^s$

$s = 1, \ldots, S_{\text{base}}$: $w_t^s \leftarrow w_t^s / W_t$

$\boldsymbol{\mu}_t' \leftarrow \sum_{s=1}^{S} w_t^s \boldsymbol{x}_t^s$

$\mathbb{V}(\boldsymbol{\mu}_t') \leftarrow \sum_{s=1}^{S_{\text{base}}} [w_t^s]^2 (\boldsymbol{x}_t^s - \boldsymbol{\mu}_t)(\boldsymbol{x}_t^s - \boldsymbol{\mu}_t)^\top$

$r_{\text{base}} \leftarrow \sqrt{\lambda_{\max}(\mathbb{V}(\boldsymbol{\mu}_t'))\ \chi_d^2(0.95)}$

$S \leftarrow S_{\text{base}} \times [r_{\text{base}}/r_{\max}]^2$

`// Filtering`

$s = 1, \ldots, S$: $\boldsymbol{x}_t^s \overset{iid}{\sim} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$

$s = 1, \ldots, S$: $w_t^s \leftarrow p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)$, $W_t \leftarrow \sum_{s=1}^{S} w_t^s$

$s = 1, \ldots, S$: $w_t^s \leftarrow w_t^s / W_t$

$\boldsymbol{x}_{t|t} \leftarrow \sum_{s=1}^{S} w_t^s \boldsymbol{x}_t^s$

$\boldsymbol{P}_{t|t} \leftarrow \sum_{s=1}^{S} w_t^s (\boldsymbol{x}_t^s - \boldsymbol{x}_{t|t})(\boldsymbol{x}_t^s - \boldsymbol{x}_{t|t})^\top$

$p(\boldsymbol{x}_t|\boldsymbol{y}_t) \leftarrow N(\boldsymbol{x}_{t|t}, \boldsymbol{\Sigma}_{t|t})$

end for

4.4.4 Alpha Divergence Kalman Filter

Algorithm 4.4 Alpha Divergence Kalman Filter (αKF)

Input: $t = 1, \ldots, T:$ $\boldsymbol{y}_t$ (observations)

$t = 1, \ldots, T:$ $\boldsymbol{F}_t$, $\boldsymbol{h}_t(\cdot)$, $\boldsymbol{Q}_t$, $\boldsymbol{R}_t$ (state-space parameters)

S (samples), $(\boldsymbol{\mu}_{0|0}, \boldsymbol{P}_{0|0})$ (initial guess)

α (alpha divergence parameter)

Output: $t = 1, \ldots, T$ $p(\boldsymbol{x}_t|\boldsymbol{y}_t) = N(\boldsymbol{x}_{t|t}, \boldsymbol{P}_{t|t})$

Initialize: $p(\boldsymbol{x}_0) = N(\boldsymbol{x}_{0|0}, \boldsymbol{P}_{0|0})$

for $t = 1, \ldots, T$ **do**

$\boldsymbol{x}_{t|t-1} \leftarrow \boldsymbol{F}_t^\top \boldsymbol{x}_{t-1|t-1}$, $\boldsymbol{P}_{t|t-1} \leftarrow \boldsymbol{F}_t^\top \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t + \boldsymbol{Q}_t$

$p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \leftarrow N(\boldsymbol{x}_{t|t-1}, \boldsymbol{P}_{t|t-1})$

$s = 1, \ldots, S$: $\boldsymbol{x}_t^s \overset{iid}{\sim} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$.

$s = 1, \ldots, S$: $w_t^s \leftarrow p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)^\alpha$, $W_t \leftarrow \sum_{s=1}^S w_t^s$

$s = 1, \ldots, S$: $w_t^s \leftarrow w_t^s / W_t$

$\boldsymbol{x}_{t|t} \leftarrow \sum_{s=1}^S w_t^s \boldsymbol{x}_t^s$

$\boldsymbol{P}_{t|t} \leftarrow \sum_{s=1}^S w_t^s (\boldsymbol{x}_t^s - \boldsymbol{x}_{t|t})(\boldsymbol{x}_t^s - \boldsymbol{x}_{t|t})^\top$

$p(\boldsymbol{x}_t|\boldsymbol{y}_t) \leftarrow N(\boldsymbol{x}_{t|t}, \boldsymbol{\Sigma}_{t|t})$

end for

4.4.5 Alpha Divergence Kalman Filter with AdaSamp

Algorithm 4.5 Alpha Divergence Kalman Filter with AdaSamp (αKF)

Input: $t = 1, \ldots, T:$ $\boldsymbol{y}_t$ (observations)

$t = 1, \ldots, T:$ $\boldsymbol{F}_t$, $\boldsymbol{h}_t(\cdot)$, $\boldsymbol{Q}_t$, $\boldsymbol{R}_t$ (state-space parameters)

$(\boldsymbol{\mu}_{0|0}, \boldsymbol{P}_{0|0})$ (initial guess)

S_{base} (initial sample size), $r_{\max}$ (error tolerance)

Output: $t = 1, \ldots, T:$ $p(\boldsymbol{x}_t|\boldsymbol{y}_t) = N(\boldsymbol{x}_{t|t}, \boldsymbol{P}_{t|t})$

Initialize: $p(\boldsymbol{x}_0) = N(\boldsymbol{x}_{0|0}, \boldsymbol{P}_{0|0})$

for $t = 1, \ldots, T$ **do**

$\boldsymbol{x}_{t|t-1} \leftarrow \boldsymbol{F}_t^\top \boldsymbol{x}_{t-1|t-1}$, $\boldsymbol{P}_{t|t-1} \leftarrow \boldsymbol{F}_t^\top \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t + \boldsymbol{Q}_t$

$p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \leftarrow N(\boldsymbol{x}_{t|t-1}, \boldsymbol{P}_{t|t-1})$

`// AdaSamp`

$s = 1, \ldots, S_{\text{base}}$: $\boldsymbol{x}_t^s \overset{iid}{\sim} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$.

$s = 1, \ldots, S_{\text{base}}$: $w_t^s \leftarrow p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)^\alpha$, $W_t \leftarrow \sum_{s=1}^S w_t^s$

$s = 1, \ldots, S_{\text{base}}$: $w_t^s \leftarrow w_t^s / W_t$

$\boldsymbol{\mu}_t' \leftarrow \sum_{s=1}^S w_t^s \boldsymbol{x}_t^s$

$\mathbb{V}(\boldsymbol{\mu}_t') \leftarrow \sum_{s=1}^{S_{\text{base}}} [w_t^s]^2 (\boldsymbol{x}_t^s - \boldsymbol{\mu}_t')(\boldsymbol{x}_t^s - \boldsymbol{\mu}_t')^\top$

$r_{\text{base}} \leftarrow \sqrt{\lambda_{\max}(\mathbb{V}(\boldsymbol{\mu}_t'))\ \chi_d^2(0.95)}$

$S \leftarrow S_{\text{base}} \times [r_{\text{base}}/r_{\max}]^2$

`// Filtering`

$s = 1, \ldots, S$: $\boldsymbol{x}_t^s \overset{iid}{\sim} p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$

$s = 1, \ldots, S$: $w_t^s \leftarrow p(\boldsymbol{y}_t|\boldsymbol{x}_t^s)^\alpha$, $W_t \leftarrow \sum_{s=1}^S w_t^s$

$s = 1, \ldots, S$: $w_t^s \leftarrow w_t^s / W_t$

$\boldsymbol{x}_{t|t} \leftarrow \sum_{s=1}^S w_t^s \boldsymbol{x}_t^s$

$\boldsymbol{P}_{t|t} \leftarrow \sum_{s=1}^S w_t^s (\boldsymbol{x}_t^s - \boldsymbol{x}_{t|t})(\boldsymbol{x}_t^s - \boldsymbol{x}_{t|t})^\top$

$p(\boldsymbol{x}_t|\boldsymbol{y}_t) \leftarrow N(\boldsymbol{x}_{t|t}, \boldsymbol{\Sigma}_{t|t})$

end for

4.4.6 Complexity Analysis

For the SKF, at every time step we start with computing the prior which is $O(d^3)$. In the inner loop the computation cost is dominated by the gradient terms which are $O(Sd^2)$. The total cost of the inner loop is then $O(ISd^2)$. Note that the gradient computation for $\mathbf{\Sigma}_t$ can be done in $O(d^2)$ if we first do the matrix-vector multiplications. The computation of natural gradients is bounded by $O(d^3)$ as we cannot avoid the matrix-matrix multiplication this time. The total cost of natural gradient computation is then $O(Id^3)$. However since $S \gg d$ typically, the per time cost is $O(ISd^2)$. The total cost of SKF is then $O(TISd^2)$.

For the MKF without AdaSamp, the prior computation is once again $O(d^3)$. Sampling from the prior is $O(Sd)$ as a single draw is $O(1)$; and computing the weights is $O(Sd^2)$ as we need to evaluate S quadratic terms. Also, computing the posterior from samples is $O(Sd^2)$ as we have S outer products. The total cost of MKF is then $O(TSd^2)$. Compared to SKF, the MKF is cheaper by a factor of I which is an important difference. The cost of αKF without AdaSamp is the same, i.e. $O(TSd^2)$ since the only difference is the dampening of the likelihood values.

For the MKF with AdaSamp, the adaptive sampling stage is $O(S_{\text{base}}d^2)$ which is once again determined solely by the covariance computation. The filtering stage is then only $O(S^{(t)}d^2)$ at time t which sums to $O(\sum_{t=1}^{T} S^{(t)}d^2)$. Since typically $\sum_{t=1}^{T} S^{(t)} \ll TS$ the AdaSamp version offers significant speed-up without notable accuracy loss, as we will show in the experiments. αKF with AdaSamp is similar.

Finally, we compare with widely used filters: EKF is only $O(Td^3)$ as it only involves a single set of matrix operations per time step. UKF also has the same cost of $O(Td^3)$ which is important as it often gives better results than EKF without increasing computation. The particle filter is once again $O(TSd^2)$ where the bottleneck is at calculating the likelihood terms. Therefore, particle filter is comparable to MKF and αKF without AdaSamp. When AdaSamp is activated, however, MKF and αKF can be significantly faster.

4.5 Experiments

We experiment with our filters and compare their performance against a number of competitors from the literature. The implemented filters are as follows:

1. EKF: Extended Kalman Filter
2. UKF: Unscented Kalman Filter
3. ENKF: Ensemble Kalman Filter [34].
4. SIR: Particle Filter with Sequential Importance Resampling.
5. SKF: Stochastic Search Kalman Filter of Algorithm 4.1.
6. MKF: Moment Matching Kalman Filter of Algorithm 4.2.
7. αKF: Alpha Divergence Kalman Filter of Algorithm 4.4.

For the applications we consider radar and sensor network target tracking problems, as well as an options pricing problem.

4.5.1 Target Tracking

The first problem we consider is target tracking. This problem arises in various settings, but here we consider two established cases: radar and sensor networks. The radar tracking problem has been a primary application area for nonlinear Kalman filtering. Wireless sensor networks are another emerging area where nonlinear filtering is useful. Driven by the advances in wireless networking, computation and micro-electro-mechanical systems (MEMS), small inexpensive sensors can be deployed in a variety of environments for many applications [27], [108].

For both problems the state-space has the form

$$\begin{aligned} \boldsymbol{x}_t &= \boldsymbol{F}\boldsymbol{x}_{t-1} + \boldsymbol{w}_t \quad , \quad \boldsymbol{w}_t \sim N(\boldsymbol{0}, \boldsymbol{Q}), \\ \boldsymbol{y}_t &= \boldsymbol{h}(\boldsymbol{x}_t) + \boldsymbol{v}_t \quad , \quad \boldsymbol{v}_t \sim N(\boldsymbol{0}, \boldsymbol{R}). \end{aligned} \tag{4.50}$$

Here, $\boldsymbol{F}$ and $\boldsymbol{Q}$ model the dynamics of target motion and are usually time-varying. On the other hand, $\boldsymbol{h}(\cdot)$ specifies the equipment that performs the measurements, and the environment and equipment based inaccuracies are represented by $\boldsymbol{R}$. In the radar setting, when the target is far away and the angle measurement noise is strong, the problem is highly nonlinear. For sensor networks, the nonlinearity is caused by the small number of active sensors (due to energy constraints) with large measurement noise (due to the attenuation in received signal) [15]. While the value of $\boldsymbol{R}$ can be determined to some extent through calibration, it is more challenging to do this for $\boldsymbol{Q}$ [75].

Our experiments are based on synthetic data using a constant velocity model in $\mathbb{R}^2$ which corresponds to the state vector vector $\boldsymbol{x}_t = [x_1,\ \dot{x_1},\ x_2,\ \dot{x_2}]^\top$; the second and fourth entries correspond to the velocity of the target in each dimension. Following [74], we set the parameters for the state variable equation to

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{F}_2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{F}_2 \end{bmatrix} \quad , \quad \boldsymbol{F}_2 = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \tag{4.51}$$

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_2 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q}_2 \end{bmatrix} \quad , \quad \boldsymbol{Q}_2 = \sigma_{CV} \begin{bmatrix} \Delta t^4/4 & \Delta t^3/2 \\ \Delta t^3/2 & \Delta t^2 \end{bmatrix}. \tag{4.52}$$

The radar measures the distance and bearing of the target via the nonlinear function $h(\cdot)$ of the target location,

$$\boldsymbol{h}(\boldsymbol{x}_t) = \left[\sqrt{x_t(1)^2 + x_t(3)^2} \quad , \quad \tan^{-1}[x_t(3)/x_t(1)]\right]^\top$$

i.e., the Cartesian-to-Polar transformation [73]. For the sensor networks problem, we will consider a scenario which uses range-only measurements from multiple sensors. This yields the model in (4.50) where $h(\cdot)$ is the measurement function such that the i-th dimension (i.e., measurement of sensor s_i) is given by

$$[\boldsymbol{h}(\boldsymbol{x}_t)]_i = \sqrt{[x_t(1) - s_i(1)]^2 + [x_t(2) - s_i(2)]^2}$$

and the length of $\boldsymbol{h}(\boldsymbol{x}_t)$ will be the number of activated sensors at time t.

We consider two types of problems: tracking with uncertain parameters and tracking with known parameters. For the case of uncertain parameters, we set the radar and sensor simulation settings as follows. First, we note that for both simulations we assume a constant measurement rate, and so $\Delta t = 1$. For radar we sweep the process noise values in Eq. (4.51) as $\sigma_{CV} \in \{10^{-3},\ 2 \times 10^{-3},\ \ldots\ ,\ 10^{-2}\}$. We generate 20 data sets for each value of σ_{CV}, yielding a total of 200 experiments. For the measurement noise we use a diagonal $\boldsymbol{R}$ with entries $\sigma_r^2 = 10^{-1}$ and $\sigma_\theta^2 = 10^{-2}$ which controls the noise of distance and bearing measurements, respectively. The initial state is $x_0 = [1000,\ 10,\ 1000,\ 10]^\top$; this distance from origin and angle noise variance results in a severely nonlinear model, making filtering quite challenging.

For sensor network simulations, we use the same constant-velocity model of Eq. (4.51) with $\sigma_{CV} = 10^{-2}$. We deploy 200 sensors and at each time point there are exactly 3 activated for range measurements. The sensors are scattered over a square field of 100×100 units sampled from a uniform distribution. For reference all sensors are shown as background in Figure 4.2. The measurement covariance matrix is $R = \sigma_R^2 \boldsymbol{I}$ where we set $\sigma_R = 20$. We set the initial state $x_0 = [1000,\ 1,\ 1000,\ 1]^\top$. With this, once again, we obtain a highly nonlinear system, albeit less severe than the radar case. We also consider the case where the generating parameters are known to the filter. In this case, we assess the performance of the filter as a function of process and measurement noise covariances. We sweep $\sigma_{\text{CV}} \in \{0.001, 0.005, 0.01, 0.05, 0.1\}$ and $\sigma_r \in \{10, 15, 20, 25, 30\}$.

The filter settings are as follows: For SKF we use 500 samples per iteration and 20 iterations, whereas the sample size is 10^4 for MKF and αKF . This way the total sample size is 10^4 for all three filters. For SIR and ENKF we also use 10^4 particles. For the αKF we set $\alpha = 0.5$; but we also separately analyze its performance as a function of α. Also, when there is parameter uncertainty, the exact value of $\boldsymbol{Q}$ is not known to the filter, in which case we simply use a scaled isotropic covariance of form $\sigma_Q^2 \boldsymbol{I}$.

Table 4.1: Radar tracking problem: Mean Square Error (MSE) of various filtering schemes as a function of process noise parameter σ_Q. The boldfaces show the best performers for small/large particle sizes.

	σ_Q				
	10^{-2}	5×10^{-2}	10^{-1}	5×10^{-1}	1
SKF	41.4100	34.6611	29.9952	42.1360	38.0507
MKF	31.3088	27.6861	29.0376	35.2422	39.2536
αKF	30.8783	27.9475	**27.4130**	31.0271	34.9420
PF	28.5429	32.3768	35.1842	44.3704	48.9767
ENKF	33.8646	35.3385	37.1761	42.2874	45.7732
EKF	33.8611	35.8086	37.7808	42.6595	45.9788
UKF	31.7528	31.8616	33.7625	41.1282	45.4806
BASE	223.5281	223.5281	223.5281	223.5281	223.5281

In Table 4.1 we show mean square error (MSE) for radar tracking as a function of the selected scale value (σ_Q). Here, the base error corresponds to the estimations based on measurements only, and its order-of-magnitude difference from filter MSE values shows the severity of nonlinearity. Comparing MSE values we see that MKF and αKF outperforms EKF and UKF for all settings of σ_Q, which shows that the Gaussian density obtained from these filters is indeed more accurate. SKF also has better results, particularly for $\sigma_Q = 10^{-1}$ but is less robust to the changes in scale value. This is due to the iterative gradient scheme employed by SKF, which could give worse results depending on parameter changes or covariance initializations. Since MKF and αKF are based on importance sampling, they do not exhibit the same sensitivity. As for PF, this algorithm also produces competitive results when $\sigma_Q = 10^{-2}$; however its performance significantly deteriorates (even more than that of SKF)

Table 4.2: Sensor network tracking problem: Mean Square Error (MSE) of various filtering schemes as a function of process noise parameter σ_Q. The boldfaces show the best performers for small/large particle sizes.

	σ_Q				
	10^{-2}	5×10^{-2}	10^{-1}	5×10^{-1}	1
SKF	10.4674	9.5812	9.5038	10.1664	10.5996
MKF	10.5572	9.2879	9.1684	9.8175	10.3307
αKF	9.9441	8.0913	**8.0623**	9.1002	9.7055
PF	9.5661	9.3464	9.4726	10.0422	10.3834
ENKF	10.6565	13.8709	11.5082	13.8403	15.2449
EKF	14.0034	13.9357	14.5161	15.5277	16.1438
UKF	11.5303	10.3639	10.2068	10.8830	11.5845

as σ_Q increases, which shows that nonparametric inference of particle filtering is more sensitive to parameter uncertainty. Lastly, ENKF has significantly worse performance than all other sampling based filters. This result is mainly because, ENKF lacks a scheme to weight the samples, and so is more sensitive to parameter uncertainties. We observe that αKF has the highest robustness to parameter changes, making it a better choice when parameters are not known and measurements are very noisy; in addition, αKF is more robust excess measurement noise, as discussed in Section 4.3.3.

Table 4.2 presents MSE results for sensor networks. Unlike the radar problem, all particle-based filters are better than EKF and UKF for all values of σ_Q. This reduced sensitivity is due to the reduced nonlinearity in the problem. The performance of SKF, MKF, and PF are similar to each other, MKF being the favorable choice for most cases. This time ENKF does a better job, since the nonlinearity is less challenging,

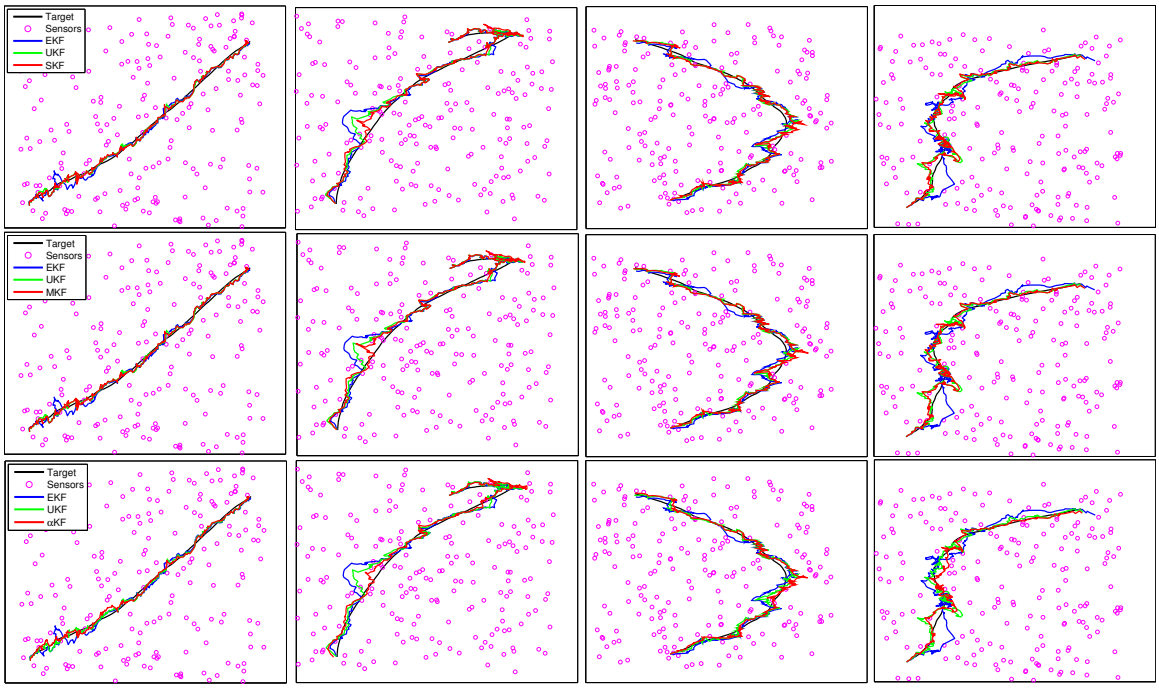


Figure 4.2: Tracks estimated by various filtering schemes in sensor network setting. Top row: Comparisons of EKF, UKF, and SKF. Middle row: EKF, UKF, and MKF. Bottom row: EKF, UKF, and αKF . In the background sensor scatterplots are given. Each plot corresponds to a square field with 100 units of side length.

yet it is still inferior to UKF. Again αKF is the best performer, and as σ_Q increases the improvement increases.

In Figure 4.2 we show qualitative tracking results with sensor networks. The top, middle, and bottom rows correspond to SKF, MKF, and αKF respectively. For each row we pick four different paths (shared across different rows) and for each plot we show the true trajectory along with EKF, UKF, and one of our filters. By visual inspection we can see that our algorithms provide more accurate tracking, which gives visual meaning to the quantitative results.

For αKF we have only considered the case when $\alpha = 0.5$. We next focus on varying α. For this experiment we set $\sigma_Q = 10^{-1}$, which corresponds to the middle column of Table 4.2. We plot the mean squared error as a function of α in Figure

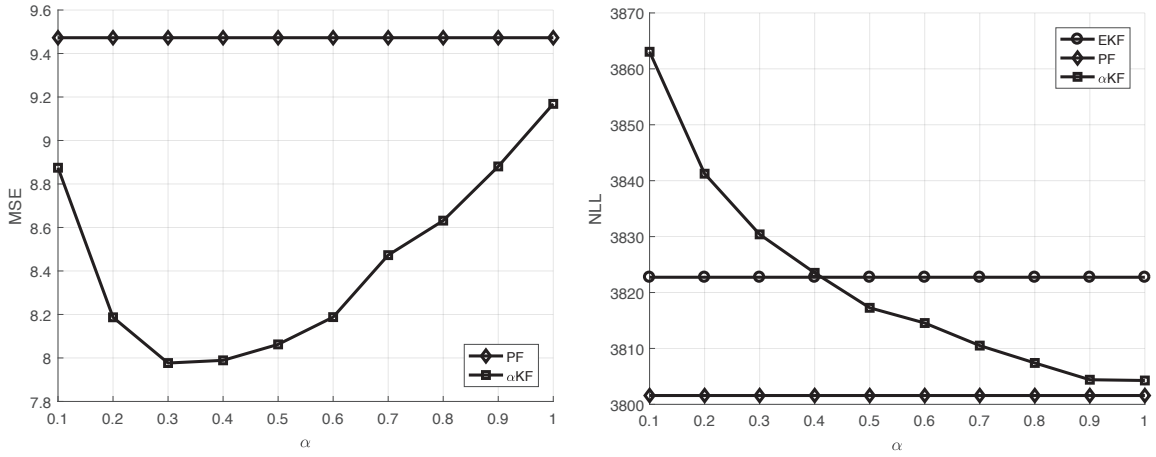


Figure 4.3: Left panel: MSE value of αKF as a function of α for the sensor network tracking problem with $\sigma_Q = 10^{-1}$. Right panel: NLL values as a function of α. For both figures, when $\alpha = 1$, αKF reduces to MKF. The performance of PF and EKF are plotted as baselines. Also, for PF and EKF the markers are only given for reference, otherwise they do not depend on α.

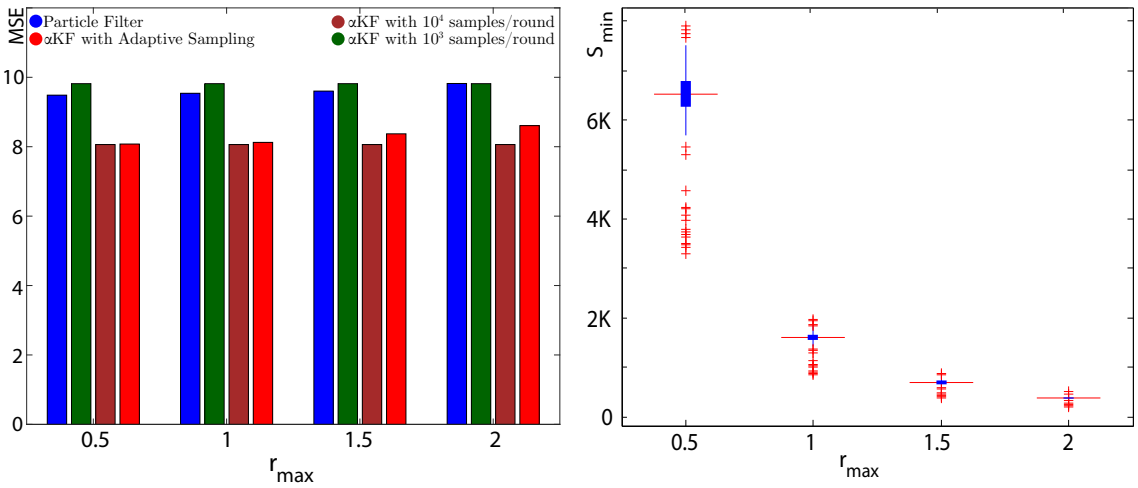
4.3. We see that low-to-mid ranges of α (i.e. $0.3 - 0.5$) give the best MSE results. This improvement is a result of lower values of α mitigating the effects of strong measurement noise. There is a trade-off, however, since choosing α too small will discard to much of information from the measurement and give poor results. This is seen for lower values of α, where decreasing the parameter degrades performance. We note that ideal values of α may differ for different sensor characteristics. Since ground truth is necessary to know the best value, simulations using known sensor characteristics can allow for selecting this parameter using a grid search as shown in Figure 4.3. Indeed, this is the main difficulty of tuning α on-the-fly; as we may not have access to such training data.

One consideration for tuning the α parameter would be to maximize the likelihood. However, this does not work in practice. In Figure 4.3 we show the negative log likelihood (NLL) for the sensor network experiment using the same setup from the left panel. We see that the lowest NLL is obtained by setting $\alpha = 1$. This is not

surprising, as this value corresponds to no dampening of the likelihood term itself. As we decrease α, NLL increases as well. In fact, while the best value of lowest MSE is $\alpha = 0.3$, its corresponding NLL is even worse than EKF. Indeed, the worse performance of EKF is tied to its sensitivity to likelihood, which in the context of machine learning is similar to overfitting. For this reason, a separate set of data would be required to tune the α parameter.

As discussed in Section 4.3.4, one can use adaptive sampling to choose the minimum possible sample size to achieve a certain confidence region radius, $r_{\max}$. We implemented adaptive sampling for αKF using an initial batch size of $S_{\text{base}} = 500$. We picked four different values of $r_{\max}$ from $\{0.5, 1, 1.5, 2\}$. Figure 4.4 displays the results for this experiment. Here we compare the MSE results as a function of $r_{\max}$ for αKF and PF for the sensor tracking problem with $\sigma_{\text{CV}} = 0.1$. Note that for SIR-PF, adaptive sampling is not a choice since all particles should be propagated, resampled, and updated at every time step. So for that we simply set the sample size as the average $S_{\min}$ for the αKF for each case. In Figure 4.4(a) we can see that the MSE performances differ very little across different cases, showing that for larger target values of $r_{\max}$ both methods can still produce accurate estimates of the true state. This is also visible in our comparison to the αKF with sample size fixed to $S = 10^4$, but performance clearly degrades for $S = 10^3$, showing the advantage of not having to set this parameter. We also see that αKF outperforms PF in all cases. On the other hand, Figure 4.4(b) shows the number of samples required to achieve a certain confidence radius. From this figure we can see the $O(1/r^2)$ decaying rate of $S_{\min}$ as implied by Eq. (4.49). Given the high accuracy in the left panel, we see that samples on the order of hundreds can be sufficient for high-quality estimates, which is an important computational gain.

We now investigate the case where the process noise parameter is known. In Figure 4.5 we show MSE as a function of σ_{CV} and σ_R. For the measurement noise, as σ_R increases, overall MSE also increases, while for process noise this trend is not

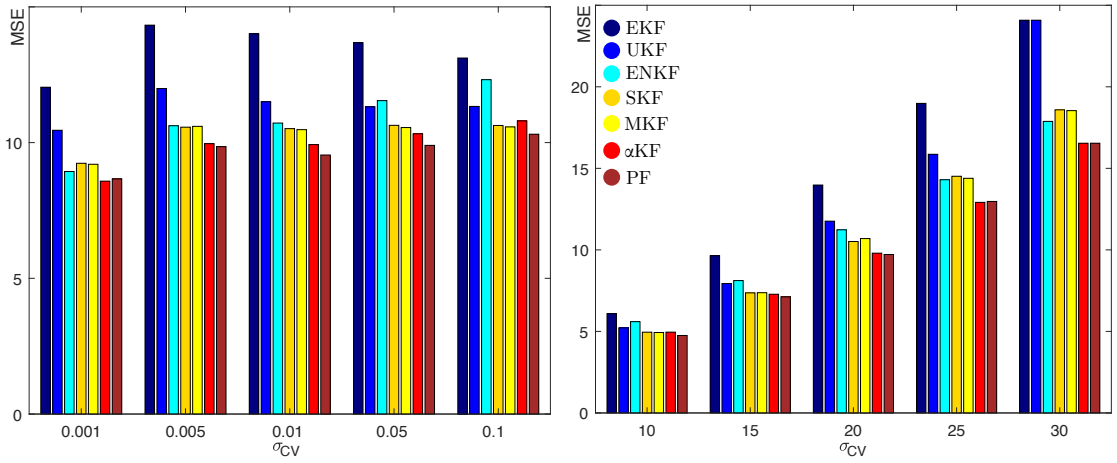


(a) MSE as a function of $r_{\max}$.

(b) Box plot of $S_{\min}$ required to achieve the given $r_{\max}$.

Figure 4.4: Mean square error and minimum sample size as a function of confidence radius $r_{\max}$.

present. For both cases we see that the particle filter gives the best result overall. This is expected, since when the parameters are known perfectly particle filters can approximate the posterior with more accuracy as they are nonparametric. However, αKF is still competitive in this setting. In fact, for several cases, such as $\sigma_{\text{CV}} = 0.001$ and $\sigma_r = 25$, performance of αKF and PF are equal, while both filters perform much better than SKF and MKF in all cases. This means αKF can be preferred over PF, since it does not require resampling. As a second observation, note that SKF/MKF perform much better than EKF/UKF, and αKF perform even better compared to the rest. This means, by minimizing different forms of divergence one can indeed get significantly better Gaussian approximations of the posterior. We also see a significant improvement for ENKF when the parameter uncertainty is removed. However, it is still not very consistent; while particle filter always produce good results, the same is clearly not the case for ENKF. Once again, this is related to the lack of sample weighting in ENKF, which causes all the samples contribute equally to the final estimate.



(a) Mean square error as a function of process noise standard deviation, σ_{CV}.

(b) Mean square error as a function of measurement noise standard deviation σ_r.

Figure 4.5: Mean square error as a function of process and measurement noise parameters, where the exact parameters are known to the filter. The legend given is shared by both figures.

4.5.2 Options Pricing

We also consider a problem from options pricing. In finance, an option is a derivative security which gives the holder a right to buy/sell (call/put option) the underlying asset at a certain price on or before a specific date. The underlying asset can be, for example, a stock. The price and date are called the strike price and expiry date respectively. The value of the option, called the premium, depends on a number of factors. Let C and P denote the call and put prices. We use σ and r to denote volatility and risk-free interest rate, respectively. The values of these variables are not directly observed and need to be estimated. Let S denote the price of the underlying asset and X the strike price. Finally, let t_m denote the time to maturity; this is the time difference between the purchase and expiry dates which is written as a fraction of a year. For example, an option which expires in two months will have $t_m = 1/6$.

Accurate pricing of options is an important problem in mathematical finance. For a European style option, the price as a function of these parameters can be modeled

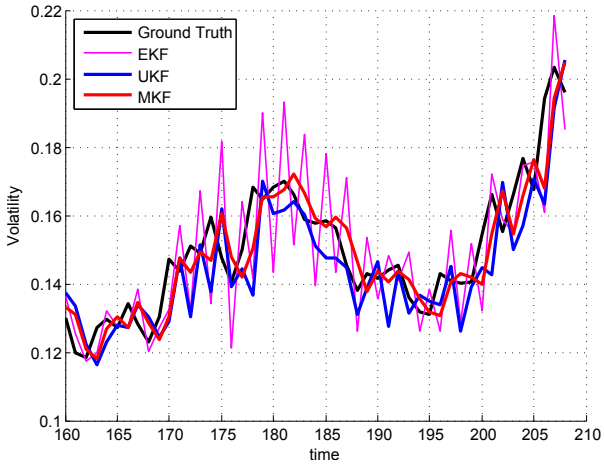


Figure 4.6: Volatility estimation performance of various filtering schemes (based on Option 1). The estimates are plotted along with the ground truth. Best viewed in color.

using the well-known Black-Scholes equation [60]

$$
\begin{aligned}
d_1 &= \frac{\log(S/X) + (r + \sigma^2/2)t_m}{\sigma\sqrt{t_m}},\ d_2 = d_1 - \sigma\sqrt{t_m} \\
C &= S\Phi(d_1) - Xe^{-rt_m}\Phi(d_2) \\
P &= -S\Phi(-d_1) + Xe^{-rt_m}\Phi(-d_2).
\end{aligned} \tag{4.53}
$$

Following the approach of [88], let $\boldsymbol{x}_t = [\sigma_t\ r_t]^\top$ be the state and $\boldsymbol{y}_t = [C_t\ P_t]^\top$ be the measurement. We therefore have the following state space representation

$$
\begin{aligned}
\boldsymbol{x}_t &= \boldsymbol{x}_{t-1} + \boldsymbol{w}_t\ , \quad \boldsymbol{w}_t \sim N(\boldsymbol{0}, \boldsymbol{Q}), \\
\boldsymbol{y}_t &= \boldsymbol{h}(\boldsymbol{x}_t) + \boldsymbol{v}_t\ , \quad \boldsymbol{v}_t \sim N(\boldsymbol{0}, \boldsymbol{R}),
\end{aligned} \tag{4.54}
$$

where the nonlinear mapping $\boldsymbol{h}(\cdot)$ is given by Eq. (4.53). In this case we model the process and measurement noises with time-invariant covariance matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. We consider two tasks: 1) predicting the one-step ahead prices, and 2) estimating the values of hidden state variables. This problem is also considered in [109] to assess the performance of particle filtering algorithms.

Table 4.3: Mean Absolute Error (MAE) values of various filtering schemes for three different call/put option pairs; calculated for $\sigma_Q = 10^{-2}$. For Option 3, EKF loses track so MAE is not reported.

		EKF	UKF	PF	SKF	MKF
Option 1 MAE	Call	0.1352	0.0788	0.0663	0.0658	0.0654
	Put	0.1528	0.0789	0.0668	0.0642	0.0654
Option 2 MAE	Call	0.0425	0.0354	0.0329	0.0312	0.0319
	Put	0.0478	0.0355	0.0340	0.0368	0.0331
Option 3 MAE	Call	-	0.2155	0.1584	0.1573	0.1586
	Put	-	0.2158	0.1579	0.1574	0.1586

We use the Black-Scholes model as the ground truth. In order to synthesize the data, we use historical values of VIX (CBOEINDEX:VIX), which measures the volatility of S&P 500 companies. From this list we pick Microsoft (NASDAQ:MSFT), Apple (NASDAQ:AAPL), and IBM (NYSE:IBM) as underlying assets and use their historical prices. The interest rate comes from a state-space model with a process noise of zero mean and variance 10^{-4}. We set $\sigma_Q = \sigma_R = 10^{-2}$. In Table 4.3 we show the next-day prediction performance of all algorithms. We can see that the prediction performance improves as we move towards MKF. This again shows the difference between Gaussian approximations of the methods we employ. For MKF and PF we used 10^3 particles; however we once again note that MKF can achieve this performance without resampling, and it can leverage adaptive sampling to reduce sample size. On the other hand, for SKF we need to use a large number of particles per iterations (around $1,000$). Even though this gives better results then EKF and UKF it is much slower than MKF and PF, and its performance can vary significantly between iterations. On the other hand, since the measurement noise is small, choosing $\alpha < 1$

for αKF does not provide improvement over MKF in this case, which is consistent with our previous discussion. Therefore $\alpha = 1$ is the best choice in this case.

Figure 4.6 shows the volatility estimation for three filters. EKF tends to over/undershoot and UKF is significantly better in that respect. However, MKF improves on these two, giving the most robust estimates. We report that the plot of SKF was similar to MKF. Also, as with the target tracking experiments, MKF has better performance than SKF, which agrees with the observation that Expectation Propagation typically outperforms Variational Inference for unimodal posteriors.

4.6 Conclusion

We have considered nonlinear Kalman filtering problem from the lens of divergence minimization. In particular we introduced three algorithms which directly minimize the forward and reverse Kullback-Leibler divergences, as well as the alpha divergence—the last divergence being a generalization of the previous two. While our algorithms are based on sampling techniques, our end goal was finding an optimal parametric distribution. We also showed how joint Gaussian assumed density filters such as the EKF and UKF optimize an approximation to the variational lower bound, meaning they only give approximately optimal solutions for the forward KL divergence.

We have conducted experiments to test the proposed methods on radar and sensor network problems, as well as options pricing. In addition to promising performance, we showed that we can obtain filters which are robust to strong measurement noise. The work here can serve as a building block for designing a class of filters which optimize a given divergence based on different choices of parametric densities. For example, it is possible to consider heavy-tailed parametric densities or multimodal densities and build dynamic filters on top of this.

4.7 Appendix to Chapter 4

4.7.1 Proof of Theorem 1

The joint Gaussian ADF corresponds to $f(\boldsymbol{x}) \approx g(\boldsymbol{x})$; this approximation is constructed from $p(\boldsymbol{y}_t|\boldsymbol{x}_t)$ in (4.8), which is Gaussian with $\boldsymbol{\mu}_{y|x} = \boldsymbol{\mu}_y + \boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}(\boldsymbol{x}_t - \boldsymbol{\mu}_x)$ and $\boldsymbol{\Sigma}_{y|x} = \boldsymbol{\Sigma}_{yy} - \boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xy}$. This yields

$$g(\boldsymbol{x}_t) = -\frac{1}{2}(\widetilde{\boldsymbol{y}}_t - \boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{x}_t)^\top \boldsymbol{R}_t^{-1}(\widetilde{\boldsymbol{y}}_t - \boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{x}_t) \tag{4.55}$$

where $\widetilde{\boldsymbol{y}}_t = \boldsymbol{y}_t - \boldsymbol{\mu}_y + \boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\mu}_x$. Note that under Eq. (4.8) we have $p(\boldsymbol{x}_t) \sim N(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_{xx})$, and let $q(\boldsymbol{x}_t) \sim N(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. Substituting $g(\boldsymbol{x}_t)$ to Eq. (4.26) the expectations are now evaluated as

$$\begin{aligned}
-\mathbb{E}_q[\log q(\boldsymbol{x}_t)] &= \frac{1}{2}\log|\boldsymbol{\Sigma}_t| \\
-\mathbb{E}_q[\log p(\boldsymbol{x}_t)] &= -\frac{1}{2}\boldsymbol{\mu}_t^\top\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\mu}_t - \frac{1}{2}\operatorname{tr}\{\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_t\} + \boldsymbol{\mu}_t\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\mu}_x \\
-\frac{1}{2}\mathbb{E}_q[g(\boldsymbol{x}_t)] &= -\frac{1}{2}\boldsymbol{\mu}_t^\top\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xy}\boldsymbol{R}_t^{-1}\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\mu}_t \\
&\quad - \frac{1}{2}\operatorname{tr}\{(\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xy}\boldsymbol{R}_t^{-1}\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1})\boldsymbol{\Sigma}_t\} \\
&\quad + \boldsymbol{\mu}_t^\top\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xy}\boldsymbol{R}_t^{-1}\widetilde{\boldsymbol{y}}_t \ .
\end{aligned} \tag{4.56}$$

The posterior parameters are found by solving $\nabla_{\boldsymbol{\mu}_t}\mathcal{L}[q(\boldsymbol{x}_t)] = 0$ and $\nabla_{\boldsymbol{\Sigma}_t}\mathcal{L}[q(\boldsymbol{x}_t)] = 0$. Differentiating the terms in Eq. (4.56) we get

$$\boldsymbol{\Sigma}_t = [\boldsymbol{\Sigma}_{xx}^{-1} + \boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xy}\boldsymbol{R}_t^{-1}\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}]^{-1} \tag{4.57}$$

$$\boldsymbol{\mu}_t = \boldsymbol{\Sigma}_t(\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xy}\boldsymbol{R}_t^{-1}\widetilde{\boldsymbol{y}}_t) \ . \tag{4.58}$$

The Matrix Inversion Lemma asserts

$$[\boldsymbol{M}_1 + \boldsymbol{M}_2\boldsymbol{M}_3\boldsymbol{M}_4]^{-1} = \boldsymbol{M}_1^{-1} - \boldsymbol{M}_1^{-1}\boldsymbol{M}_2[\boldsymbol{M}_3^{-1} + \boldsymbol{M}_4\boldsymbol{M}_1^{-1}\boldsymbol{M}_2]^{-1}\boldsymbol{M}_4\boldsymbol{M}_1^{-1} \ .$$

Applying this to Eq. (4.57) we obtain

$$\begin{aligned}
\boldsymbol{\Sigma}_t &= \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xy}(\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xy} + \boldsymbol{R}_t)\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xx} \\
&= \boldsymbol{\Sigma}_{xx} - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx} \ .
\end{aligned} \tag{4.59}$$

Substituting Eq. (4.59) into Eq. (4.58) and expanding we get

$$\begin{aligned}\boldsymbol{\mu}_t &= \boldsymbol{\mu}_x - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy}\boldsymbol{R}_t^{-1}\widetilde{\boldsymbol{y}}_t - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\Sigma}_{xy}\boldsymbol{R}_t^{-1}\widetilde{\boldsymbol{y}}_t \\ &= \boldsymbol{\mu}_x - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\mu}_x + (\boldsymbol{I} - \boldsymbol{\Sigma}_{xx}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1})\boldsymbol{\Sigma}_{xy}\boldsymbol{R}_t^{-1}\widetilde{\boldsymbol{y}}_t \\ &= \boldsymbol{\mu}_x - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1}\boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\widetilde{\boldsymbol{y}}_t \\ &= \boldsymbol{\mu}_x + \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}(\boldsymbol{y}_t - \boldsymbol{\mu}_y) \ . \end{aligned} \tag{4.60}$$

Note the third line follows from the identity $\boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1} = (\boldsymbol{I} - \boldsymbol{\Sigma}_{xy}\boldsymbol{\Sigma}_{yy}^{-1}\boldsymbol{\Sigma}_{yx}\boldsymbol{\Sigma}_{xx}^{-1})\boldsymbol{\Sigma}_{xy}\boldsymbol{R}_t^{-1}$ which can be verified with straightforward manipulation. Matching the terms in Eq. (4.12) with Eq. (4.60) and Eq. (4.59) we obtain the updates in Eq. (4.11). ■

4.7.2 Proof of Corollary 2

The proof is similar to that of Theorem 1, therefore we highlight the key points. We change the notation to $p(\boldsymbol{x}_t) \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $q(\boldsymbol{x}_t) \sim N(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$. We employ a first-order Taylor series expansion around prior mean: $\boldsymbol{h}(\boldsymbol{x}_t) \approx \boldsymbol{h}(\boldsymbol{\mu}) + \boldsymbol{H}_t(\boldsymbol{\mu})(\boldsymbol{x}_t - \boldsymbol{\mu})$ where $\boldsymbol{H}_t(\boldsymbol{\mu})$ is the Jacobian. Define $\widetilde{\boldsymbol{y}}_t = \boldsymbol{y}_t - \boldsymbol{h}(\boldsymbol{\mu}) + \boldsymbol{H}_t(\boldsymbol{\mu})\boldsymbol{\mu}$. Plugging these into the variational lower bound in Eq. (4.26) and differentiating we obtain

$$\boldsymbol{\Sigma}_t = \left(\boldsymbol{\Sigma}^{-1} + \boldsymbol{H}_t^\top \boldsymbol{R}_t^{-1}\boldsymbol{H}_t\right)^{-1} \ , \tag{4.61}$$

$$\boldsymbol{\mu}_t = \boldsymbol{\Sigma}_t \left(\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{H}_t^\top \boldsymbol{R}_t^{-1}\widetilde{\boldsymbol{y}}_t\right) \ . \tag{4.62}$$

Once again, using the matrix inversion lemma we get

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma} - \boldsymbol{K}_t\boldsymbol{S}_t\boldsymbol{K}_t^\top \ , \tag{4.63}$$

where $\boldsymbol{S}_t = \boldsymbol{H}_t\boldsymbol{\Sigma}\boldsymbol{H}_t^\top + \boldsymbol{R}_t$ and $\boldsymbol{K}_t = \boldsymbol{\Sigma}\boldsymbol{H}_t^\top {\boldsymbol{S}_t}^{-1}$. Plugging Eq. (4.63) in Eq. (4.62) and expanding the multiplication we get

$$\begin{aligned}\boldsymbol{\mu}_t &= \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{H}_t^\top \boldsymbol{R}_t^{-1}\widetilde{\boldsymbol{y}}_t - \boldsymbol{K}_t\boldsymbol{H}_t^\top\boldsymbol{\mu} - \boldsymbol{K}_t\boldsymbol{H}_t\boldsymbol{\Sigma}\boldsymbol{H}_t^\top \boldsymbol{R}_t^{-1}\widetilde{\boldsymbol{y}}_t \\ &= \boldsymbol{\mu} - \boldsymbol{K}_t\boldsymbol{H}_t^\top\boldsymbol{\mu} + (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H}_t)\boldsymbol{\Sigma}\boldsymbol{H}_t^\top \boldsymbol{R}_t^{-1}\widetilde{\boldsymbol{y}} \\ &= \boldsymbol{\mu} - \boldsymbol{K}_t\boldsymbol{H}_t^\top\boldsymbol{\mu} + \boldsymbol{K}_t\widetilde{\boldsymbol{y}}_t \\ &= \boldsymbol{\mu} + \boldsymbol{K}_t(\boldsymbol{y}_t - \boldsymbol{h}_t(\boldsymbol{\mu})) \end{aligned} \tag{4.64}$$

where the first and third lines utilize the identities $\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma} - \boldsymbol{K}_t\boldsymbol{H}_t\boldsymbol{\Sigma}$ and $\boldsymbol{K}_t = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H}_t)\boldsymbol{\Sigma}\boldsymbol{H}_t^\top\boldsymbol{R}_t^{-1}$ respectively. We see that Eq. (4.64) and Eq. (4.63) correspond to the EKF update equations. ■

Part III

AUC Maximization

Chapter 5

Mini-Batch AUC Maximization

5.1 Introduction

The focus of the previous chapters have been on time series analysis: The Collaborative Kalman Filter of Chapter 2 is a method to predict the missing values of a continuous or discrete valued dyadic process. Online Forecasting Matrix Factorization in Chapter 3 is concerned with forecasting the future values of sparse and high dimensional time series. On the other hand, the previous Chapter 4 is about the more general nonlinear Kalman filtering problem. In this chapter we turn our attention to a somewhat more traditional machine learning problem, which is within the binary classification framework. In particular, given a set of positive and negative inputs, we are concerned with building a scoring system, such that the positive samples are ranked higher than the negative ones.

Historically the problem has been studied extensively in signal processing literature [92], in particular for radars, as missing the presence of a target (miss detection) could have dire consequences. In such a setting, a received signal is assigned a score, which is then compared against a threshold to rule if a target is present. The receiver operating characteristics (ROC) curve plots the ratio of true positives (detection) to false positives (false alarm) as a function of this threshold, and pro-

vides information about the behavior of the system. The area under the ROC curve (AUC) is a threshold-independent metric which measures the fraction of times a positive instance is ranked higher than a negative one. Therefore, this problem is also known as the bipartite ranking or AUC maximization problem; the latter being the title of this chapter. This has a wide variety of applications such as, for example, imaging [77], dictionary learning [26], signal detection [62], and noise-enhanced detection [24], [25], [10].

The AUC maximization problem is not limited to traditional applications. In fact, is has an important place in modern machine learning. Important subfields include cost-sensitive and imbalanced learning [56], [22], [76]. In the latter, one is given a dataset where the number of negative samples is dominating. This means a classifier which predicts all incoming instances to be negative will have very high prediction accuracy. On the other hand, it will have an AUC of zero. This is worse than random guessing, which would give 0.5, and so the AUC can be a better choice for performance metric, and devising methods to achieve higher AUC is a meaningful goal. For this reason, the AUC metric is heavily used for website ad click prediction problems [82], where only a very small fraction of web history contains ads clicked by visitors. In this case, a system with high AUC is the one which can distinguish the ads that are relevant for a user, whereas a simple classifier that maximizes prediction accuracy may simply predict all ads as uninteresting.

Given that AUC is the primary performance measure for many problems, it is useful to devise algorithms that directly optimize it during the training phase. AUC maximization has been studied within the context of well-known machine learning methods, such as support vector machines [17], boosting [38], and decision trees [37]. However, most of these traditional approaches do not scale with data size. This is because the AUC is defined over the positive/negative instance pairs which has a growth rate of $O(N_+N_-)$. Moreover, the AUC itself is a sum of indicator functions, and its direct optimization is NP-hard.

Recent research in this direction increasingly focuses on convex surrogate loss functions to represent the AUC. This enables one to use stochastic gradient methods to efficiently learn a ranking function [117]. The first work in this direction is [118], where an online AUC maximization method based on proxy hinge loss is proposed. Later, [39] uses the pairwise squared loss function, which eliminates the need for buffering previous instances; [32] proposes adaptive subgradient methods which can handle sparse inputs, while [58], [59] consider the nonlinear AUC maximization problem using kernel and multiple-kernel methods. Most recently, [31] focuses on scalable kernel methods.

While these approaches can significantly increase scalability, for very large datasets their sequential nature can still be problematic. One widely used technique—particularly for training deep neural networks on large datasets [42]—is processing data in mini-batches. This is different than the previous chapters in this thesis, where the focus was on sequential processing as the data was arriving as a stream. Here we relax this assumption and consider data available in batch. In this case, AUC maximization using mini-batches of data is desirable.

In this chapter we propose a novel algorithm for fast AUC maximization. Our approach, called Mini-Batch AUC Maximization (MBA) is based on a convex relaxation of the AUC function. However instead of using stochastic gradients, it uses the first and second order U-statistics of pairwise differences. A distinctive feature of our approach is it being learning-rate free, contrary to mini-batch gradient descent methods. This is important, as tuning the step size *a priori* is a difficult task, and generic approaches such as cross-validation are inefficient when the dataset is large.

One of the main challenges of AUC optimization is, even if a convex relaxation is applied, the resulting problem is still defined over pairs of positive/negative samples, whose optimization has a sample cost of $O(N_+N_-)$. This is prohibitively large even for moderate datasets. Since mini-batch optimization is based on sub-sampling, it is important to understand the behavior of MBA as a function of sample size. Our

theoretical analysis reveals that, the solution returned by MBA concentrates around the all-pairs solution exponentially fast. Unlike previous work, our proofs are based on the more recent results on matrix concentration [107], and quite straightforward. This shows how the U-processes —which were shown to be useful for ranking problems [30]—can be utilized to obtain a scalable mini-batch algorithm.

This chapter is organized as follows. In Section 5.2 we start with an overview of the bipartite ranking problem and our learning setup. In Section 5.3 we develop the MBA algorithm and in Section 5.4 we present the theoretical analysis. Section 5.6 contains extensive experiments including a simulation study, fifteen datasets from UCI/LIBSVM repositories, and three large-scale commercial web click data. We conclude in Section 5.7.

5.2 Background

A widely studied problem in machine learning is binary classification, in which for each given input $\boldsymbol{x} \in \mathcal{X}$ there is a a corresponding label $y \in \mathcal{Y} = \{+, -\}$. In the learning setup, a set of labeled data is provided for training, and the aim is to make accurate predictions on the unknown inputs. As usual in such settings, we assume that the samples provided for training are iid with the following unknown distributions [1]

$$[\boldsymbol{X} \mid y = +] \sim \mathcal{P}^+ \quad , \quad [\boldsymbol{X} \mid y = -] \sim \mathcal{P}^- \tag{5.1}$$

Since the distributions are unknown, a model $f(\cdot)$ is fitted to the training data using a specific loss function, such as cross-entropy loss for logistic regression and hinge loss for support vector machine. Here, the loss function we focus on is the Area Under Receiver Operating Characteristics (ROC) Curve, which is abbreviated

[1] Here we consider the case with no label noise, however the modification to that case is straightforward.

as AUC. Its functional form is given by

$$\text{AUC} = \mathbb{E}_{\substack{\boldsymbol{x}^+ \sim \mathcal{P}^+ \\ \boldsymbol{x}^- \sim \mathcal{P}^-}} \Big[\mathbb{1}\{f(\boldsymbol{x}^+) - f(\boldsymbol{x}^-) > 0\} \Big] \tag{5.2}$$

which measures how well $f(\cdot)$ ranks a positive sample higher than a negative one. Eq. (5.2) shows that the AUC is defined as the expectation of an indicator function. This means the corresponding empirical risk function will be a sum of indicators, resulting in an NP-hard optimization problem. For this reason we first discuss a relaxation of the original problem using convex surrogate loss functions.

Replacing $\mathbb{1}[f(\boldsymbol{x}^+) - f(\boldsymbol{x}^-) > 0]$ in Eq. (5.2) with the pairwise convex surrogate loss $\phi(\boldsymbol{x}^+, \boldsymbol{x}^-) = \phi(f(\boldsymbol{x}^+) - f(\boldsymbol{x}^-))$, the aim now becomes to minimize the ϕ-risk [87]

$$R_\phi(f) = \mathbb{E}_{\substack{\boldsymbol{x}^+ \sim \mathcal{P}^+ \\ \boldsymbol{x}^- \sim \mathcal{P}^-}} \left[\phi(f(\boldsymbol{x}^+) - f(\boldsymbol{x}^-)) \right]. \tag{5.3}$$

This is the Bayes risk of the scoring function [9]. There are many possible choices for surrogate function; some common choices are the pairwise squared loss (PSL), pairwise hinge loss (PHL), pairwise exponential loss (PEL), and pairwise logistic loss (PLL) [40]:

$$\begin{aligned} \phi_{\text{PSL}}(t) &= (1-t)^2, \;\; \phi_{\text{PHL}}(t) = \max(0,\; 1-t), \\ \phi_{\text{PEL}}(t) &= \exp(-t), \;\; \phi_{\text{PLL}}(t) = \log(1 + \exp(-t)), \end{aligned} \tag{5.4}$$

where $t := f(\boldsymbol{x}^+) - f(\boldsymbol{x}^-)$ is the pairwise scoring difference. Among the recent works on AUC optimization, [118] and [67] use PHL, whereas [39] and [32] focus on PSL. On the other hand, all these studies are focused on deriving a stochastic gradient-based algorithm. Our choice here is the PSL function for two reasons: (i) its consistency with the original AUC loss is proven [40], and (ii) the structure of PSL allows for a mini-batch algorithm, for which theoretical guarantees can be derived. Unlike stochastic gradient methods, this formulation is learning rate-free, which quite notably increases its practicality.

We now take one further step and assume that the scoring function is linear in the original input space; however we will discuss nonlinear extensions in Section 5.3.

In this case we have the further simplification $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ and the ϕ-risk becomes

$$\begin{aligned} R_\phi(f) &= \mathbb{E}_{\substack{\boldsymbol{x}^+ \sim \mathcal{P}^+ \\ \boldsymbol{x}^- \sim \mathcal{P}^-}} \left[\left(\, 1 - \boldsymbol{w}^\top (\boldsymbol{x}^+ - \boldsymbol{x}^-) \, \right)^2 \right] \\ &= 1 - 2\boldsymbol{w}^\top \mathbb{E}\left[(\boldsymbol{x}^+ - \boldsymbol{x}^-) \right] \\ &\qquad + \boldsymbol{w}^\top \mathbb{E}\left[(\boldsymbol{x}^+ - \boldsymbol{x}^-)(\boldsymbol{x}^+ - \boldsymbol{x}^-)^\top \right] \boldsymbol{w} \\ &= 1 - 2\boldsymbol{w}^\top \boldsymbol{\mu} + \boldsymbol{w}^\top \boldsymbol{\Sigma} \boldsymbol{w}, \end{aligned} \tag{5.5}$$

where we defined $\boldsymbol{x}_{\text{diff}} := (\boldsymbol{x}^+ - \boldsymbol{x}^-)$, and $\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{x}_{\text{diff}}]$ and $\boldsymbol{\Sigma} = \mathbb{E}[\boldsymbol{x}_{\text{diff}}\, \boldsymbol{x}_{\text{diff}}^\top]$. Note that $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ characterize the first and second moments of $\boldsymbol{x}_{\text{diff}}$—the pairwise difference. This is important because the quality of bipartite ranking does not directly depend on the positive and negative inputs, but the differences between them. This observation will form the basis of our mini-batch algorithm in Section 5.3. Finally, by definition, $\boldsymbol{\Sigma}$ is positive semi-definite and when it is positive definite, there is a unique $\boldsymbol{w}^\star$ that achieves the minimum $R_\phi(f)$.

At this point, optimizing either one of the objectives in Eq. (5.2) or Eq. (5.5) would require knowledge of $\mathcal{P}^+$ and $\mathcal{P}^-$, which is unavailable. Instead we are given iid samples from these distributions, as we mentioned in the beginning. The task is to learn a score function which should yield high AUC on the test data. Here the corresponding empirical AUC metric is

$$\text{AUC} = \sum_{i=1}^{N_+} \sum_{j=1}^{N_-} \mathbb{1}\{ f(\boldsymbol{x}_i^+) - f(\boldsymbol{x}_j^-) > 0 \} \tag{5.6}$$

which replaces the expectation in Eq. (5.2) with sample-base average. As mentioned before, direct optimization of the empirical AUC is NP-hard as it is a sum of indicators, furthermore the sum itself contains pairs that grow quadratically with the training data. To sidestep this difficulty, a surrogate loss function $\phi(\cdot)$ can be chosen to replace the Eq. (5.6), as we did for (5.2). In particular, the empirical ϕ-risk (c.f. Eq. (5.3)) has the following general form

$$\widehat{R}_\phi(f) = \frac{1}{2N_+N_-} \sum_{i=1}^{N_+} \sum_{j=1}^{N_-} \phi(f(\boldsymbol{x}_i^+) - f(\boldsymbol{x}_j^-)) \; . \tag{5.7}$$

Based on this final equation we will derive the MBA algorithm in the next section.

5.3 Mini-Batch AUC Maximization

Using the pairwise squared loss function we obtained a convex optimization problem in place of the original NP-hard problem; however, Eq. (5.3) still cannot be used as it relies on the data generating distribution $\mathcal{P}$. In practical settings, we are only given a set of positive and negative instances sampled from $\mathcal{P}$, written $\mathcal{S}^+ = \{\boldsymbol{x}_1^+, \ldots, \boldsymbol{x}_{N_+}^+\}$, $\mathcal{S}^- = \{\boldsymbol{x}_1^-, \ldots, \boldsymbol{x}_{N_-}^-\}$. We therefore substituted the empirical risk in Eq. (5.7) which can be more easily optimized. The term $N := N_+ N_-$ corresponds to the total number of pairs in the data. Similar to the ϕ-risk, optimizing the empirical risk yields a convex problem; but the number of pairs grows quadratic in the number of data points. Therefore, even for moderate datasets, minimizing the empirical risk in Eq. (5.7) becomes intractable.

We now consider a linear scoring function of form $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$ which allows us to write Eq. (5.7) as

$$\begin{aligned}\widehat{R}_\phi(\boldsymbol{w}) = -\boldsymbol{w}^\top \left[\frac{1}{N_+ N_-} \sum_{i=1}^{N_+} \sum_{j=1}^{N_-} (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)\right] \\ + \frac{1}{2} \boldsymbol{w}^\top \left[\frac{1}{N_+ N_-} \sum_{i=1}^{N_+} \sum_{j=1}^{N_-} (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)(\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)^\top\right] \boldsymbol{w}\end{aligned} \tag{5.8}$$

where we can define

$$\begin{aligned}\boldsymbol{\mu}_N &= \frac{1}{N_+ N_-} \sum_{i=1}^{N_+} \sum_{j=1}^{N_-} (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-) \\ \boldsymbol{\Sigma}_N &= \frac{1}{N_+ N_-} \sum_{i=1}^{N_+} \sum_{j=1}^{N_-} (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)\ (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)^\top.\end{aligned} \tag{5.9}$$

The variables in Eq. (5.9) are sample-based approximations of the first and second moments of $\boldsymbol{x}_{\text{diff}}$, i.e. they are approximations of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ in Eq. (5.3). Overall, the

optimization problem to be solved is

$$\boldsymbol{w}_N = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_N \boldsymbol{w} - \boldsymbol{w}^\top \boldsymbol{\mu}_N + R_{\mathrm{EL}}(\boldsymbol{w}), \tag{5.10}$$

where $R_{\mathrm{EL}}(\boldsymbol{w}) := \lambda_1 \|\boldsymbol{w}\|_1 + (1/2)\lambda_2 \|\boldsymbol{w}\|_2^2$ is the elastic net regularizer [120], which we add to prevent overfitting. Note that, unlike Eq. (5.3), there is always a unique optimum since the elastic net penalty makes the objective strictly convex. In addition, this regularizer encourages a solution which combines small ℓ_2 norm with sparsity. By substituting appropriate values for λ_1 and λ_2 we also recover ridge and lasso regression. In Section 5.6 we report experiment results for all three cases.

Since it is impractical to use all N samples, we propose to use mini-batches to obtain estimates of the moments. This is a simple process which only requires the computation of U-statistics. Note that, given a parameter θ and symmetric measurable function h which satisfies $\theta = h(X_1, \ldots, X_m)$, the corresponding U-statistic is given by

$$U_n = \binom{n}{m}^{-1} \sum_{C_{n,m}} h(X_1, \ldots, X_n), \tag{5.11}$$

where $C_{n,m}$ is the set of all length-m combinations with increasing indices. As the name implies the U-statistics are unbiased, so $\theta = \mathbb{E}[U_n]$, and provide best unbiased estimators [110]. On the other hand, a U-statistic of the second moment matrix $\boldsymbol{\Sigma}_N$ also provides a building block to get an exponential concentration bound [107].[2] Our theoretical analysis will use this property.

We now describe the MBA algorithm. Let T be the total number of rounds. At round t we sample B positive and B negative samples from the entire population with replacement. Let $\mathcal{S}_t^+$ and $\mathcal{S}_t^-$ be the arrays of sample indices and let $\mathcal{S}_t$ be the array of pairs stored as tuples of the form $(\mathcal{S}_t^+(i), \mathcal{S}_t^-(i))$—note that we do not form the Cartesian product. The expressions for U-statistics of the first and second moments

[2]Note that for a *given* training set with N pairs, the quantity $\boldsymbol{\Sigma}_N$ is an unknown constant.

simplify from Eq. (5.11) as

$$\begin{aligned}\boldsymbol{\mu}_t &:= \frac{1}{B} \sum_{(i,j)\in\mathcal{S}_t} (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-) \\ \boldsymbol{\Sigma}_t &:= \frac{1}{B} \sum_{(i,j)\in\mathcal{S}_t} (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)\ (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)^\top. \end{aligned} \tag{5.12}$$

Finally let $S = BT$ denote the total number of pairs sampled by our algorithm. We also introduce the notation $\mathcal{S}_{1:T}$ for the entire array of pairs sampled during all rounds. The overall moment approximations are therefore

$$\begin{aligned}\boldsymbol{\mu}_S &:= \frac{1}{BT} \sum_{(i,j)\in\mathcal{S}_{1:T}} (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-) \\ \boldsymbol{\Sigma}_S &:= \frac{1}{BT} \sum_{(i,j)\in\mathcal{S}_{1:T}} (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)\ (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)^\top, \end{aligned} \tag{5.13}$$

and the optimization problem constructed by MBA is

$$\boldsymbol{w}_S = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_S \boldsymbol{w} - \boldsymbol{w}^\top \boldsymbol{\mu}_S + R_{\mathrm{EL}}(\boldsymbol{w}). \tag{5.14}$$

This is the function MBA aims to construct and solve, which itself is an approximation of the global risk minimization problem in Eq. (5.3). On the other hand, stochastic gradient-based approaches make local gradient approximations of the global function and seek a solution that way. As we will show in the experiments, this is an important difference and MBA can find better solutions since it constructs a global problem first. MBA is summarized in Algorithm 5.1.

Mini-batch optimization is heavily employed in machine learning, including training of deep neural networks [42] and scalable Bayesian inference [55]. The main benefit of using mini-batches is, it is significantly faster compared to the sequential approach. Online methods for optimizing AUC, however, require sequential processing, as the parameters are updated per input. This is the main reason MBA offers a significant improvement in speed. In addition to this, MBA offers several other advantages. Since sampling pairs and computing U-statistics is an isolated process, MBA can easily be distributed across machines, which can work in an asynchronous manner.

Therefore MBA is suitable for cluster computing. Secondly, streaming and/or non-stationary data processing can be incorporated into the MBA framework, as it can process streams as blocks and give larger weights to more recent ones.

Remark: From an algorithmic perspective, as far as the sample size is concerned the only important parameter is S; we introduced B and T to further manage computational resources. For instance when S samples are too large to fit into memory we can instead use B samples in T rounds. Alternatively, S samples can be obtained by T machines with B samples per machine in parallel. In any case, we will have S uniform pairs sampled from the entire set.

5.4 Theoretical Analysis

Solving the regularized empirical risk minimization problem in Eq. (5.10) requires processing N pairwise samples. As this number grows quadratically with the number of positive and negative samples, it is often not possible to do this exactly. The proposed MBA addresses this problem by approximating the N-pair problem with an S-pair one, where $S \ll N$ samples are collected in mini-batches. This results in the problem in Eq. (5.14). Clearly the success of this approach depends on how well the second problem can approximate the first, while keeping $S \ll N$. In this section we provide a rigorous argument in favor of MBA.

Formally, given N samples we can solve the following regularized empirical risk minimization problem

$$\boldsymbol{w}_N = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_N \boldsymbol{w} - \boldsymbol{w}^\top \boldsymbol{\mu}_N + R_{\mathrm{EL}}(\boldsymbol{w})$$

however as N can be very large we instead solve

$$\boldsymbol{w}_S = \arg\min_{\boldsymbol{w}} \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_S \boldsymbol{w} - \boldsymbol{w}^\top \boldsymbol{\mu}_S + R_{\mathrm{EL}}(\boldsymbol{w})$$

where S uniform samples are used. We would like to show that the two solutions are close to each other, while $S \ll N$. We next derive a bound for the closeness of

the solutions. A natural measure of this is the Euclidean distance $d(\boldsymbol{w}_N - \boldsymbol{w}_S) = \|\boldsymbol{w}_N - \boldsymbol{w}_S\|_2^2$ which is also used by recent work on matrix sketching [91]. Therefore, Theorem 2 of this section gives a high probability bound in terms of the Euclidean distance. This is done by bounding the difference between final costs $|\mathcal{L}_N(\boldsymbol{w}_N) - \mathcal{L}_N(\boldsymbol{w}_S)|$, which is a measure regret [23].

We define the following two functions for convenience,

$$\begin{aligned}\mathcal{L}_N(\boldsymbol{w}) &= \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_N \boldsymbol{w}^\top - \boldsymbol{w}^\top \boldsymbol{\mu}_N + \lambda_1 \|\boldsymbol{w}\|_1 + \frac{\lambda_2}{2}\|\boldsymbol{w}\|_2^2 \\ \mathcal{L}_S(\boldsymbol{w}) &= \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_S \boldsymbol{w}^\top - \boldsymbol{w}^\top \boldsymbol{\mu}_S + \lambda_1 \|\boldsymbol{w}\|_1 + \frac{\lambda_2}{2}\|\boldsymbol{w}\|_2^2 \end{aligned} \tag{5.15}$$

and let $\boldsymbol{w}_N$ and $\boldsymbol{w}_S$ denote the unique minimizers as in Eqs. (5.10) and (5.14). Since $\mathcal{L}_N(\boldsymbol{w})$ is strictly convex, for a fixed δ there exists an ϵ such that

$$|\mathcal{L}_N(\boldsymbol{w}_N) - \mathcal{L}_N(\boldsymbol{w}_S)| \leq \epsilon \implies \|\boldsymbol{w}_N - \boldsymbol{w}_S\|_2 \leq \delta. \tag{5.16}$$

Clearly, ϵ is the infimum of $\mathcal{L}_N(\cdot)$ over the circle centered at $\boldsymbol{w}_N$ with radius δ. Consequently, one can focus on bounding the objective function, and this will yield the desired bound on the solutions. We now introduce ℓ_2-norm bounds on the data and weight vectors: For any given input we assume that $\|\boldsymbol{x}\|_2^2 \leq R_x$;[3] next, define the upper bound on weights such that $\max\{\|\boldsymbol{w}_N\|_2^2, \|\boldsymbol{w}_S\|_2^2\} \leq R_w$. Note here that $R_w < \infty$ is guaranteed by the ℓ_2 regularization of the elastic net. We define the difference between the two objective functions in Eq. (5.15) to be

$$\Delta(\boldsymbol{w}) := \mathcal{L}_S(\boldsymbol{w}) - \mathcal{L}_N(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^\top(\boldsymbol{\Sigma}_S - \boldsymbol{\Sigma}_N)\boldsymbol{w} + \boldsymbol{w}^\top(\boldsymbol{\mu}_N - \boldsymbol{\mu}_S)\ . \tag{5.17}$$

Two important quantities of interest are

$$\begin{aligned}\Delta_{\boldsymbol{\Sigma}} &= \boldsymbol{\Sigma}_S - \boldsymbol{\Sigma}_N \\ \Delta_\sigma &= (\boldsymbol{w}_1 - \boldsymbol{w}_2)^\top(\boldsymbol{\mu}_N - \boldsymbol{\mu}_S)\ . \end{aligned} \tag{5.18}$$

[3]This is a mild assumption since training data is typically normalized.

where Δ_σ is defined for *arbitrary* $\boldsymbol{w}_1, \boldsymbol{w}_2$ such that $\|\boldsymbol{w}_1\|_2^2, \|\boldsymbol{w}_2\|_2^2 \leq R_w$. Defining $\Delta_{\boldsymbol{w}} := \boldsymbol{w}_1 - \boldsymbol{w}_2$ we write $\Delta_\sigma = \Delta_{\boldsymbol{w}}^\top(\boldsymbol{\mu}_N - \boldsymbol{\mu}_S)$ Importantly, the difference $|\mathcal{L}_N(\boldsymbol{w}_N) - \mathcal{L}_N(\boldsymbol{w}_S)|$ can be bounded through $\Delta(\boldsymbol{w})$, which in turn can be bounded through $\Delta_{\boldsymbol{\Sigma}}$ and Δ_σ. The following lemma provides two concentration bounds for $\Delta_{\boldsymbol{\Sigma}}$ and Δ_σ.

Lemma 1: Let $\|\Delta_{\boldsymbol{\Sigma}}\|_2$ and $|\Delta_\sigma|$ denote the spectral and ℓ_1 norms respectively. For $\gamma > 0$ and sample size S,

$$\text{(i)} \quad P(\|\Delta_{\boldsymbol{\Sigma}}\|_2 > \gamma) \leq 2d \exp\left\{-S\frac{\gamma^2}{4R_x\|\boldsymbol{\Sigma}_N\|_2 + (8/3)\gamma R_x}\right\}$$

$$\text{(ii)} \quad P(|\Delta_\sigma| > \gamma) \leq 2 \exp\left\{-S\frac{\gamma^2}{4R_w\|\boldsymbol{\Sigma}_N\|_2 + (8/3)\gamma\sqrt{R_x R_w}}\right\}$$

Proof: We will use the shorthand $\Delta_{\boldsymbol{w}} = \boldsymbol{w}_N - \boldsymbol{w}_S$, $\Delta_{\boldsymbol{\mu}} = \boldsymbol{\mu}_N - \boldsymbol{\mu}_S$, and $\Delta\boldsymbol{x}_s = (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)$ where $\mathcal{S}[s] = (i, j)$. We also recall the Bernstein inequality for a $d \times d$ symmetric, random matrix $\boldsymbol{Z} = \sum_s \boldsymbol{E}_s$ and threshold γ

$$P[\|\boldsymbol{Z}\|_2 > \gamma] \leq 2d \exp\left(\frac{-\gamma^2/2}{\mathbb{V}(\boldsymbol{Z}) + L\gamma/3}\right) \tag{5.19}$$

where $\|\boldsymbol{E}_s\| \leq L$. The scalar version is recovered by setting $d = 1$.

(i) This part follows the argument for the sample covariance estimator in [107]. For the matrix we can write $\Delta_{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}_S - \boldsymbol{\Sigma}_N = \sum_{s\in\mathcal{S}} \frac{1}{S}[\Delta\boldsymbol{x}_s\Delta\boldsymbol{x}_s^\top - \boldsymbol{\Sigma}_N]$. We denote each summand by $\boldsymbol{E}_s = \frac{1}{S}[\Delta\boldsymbol{x}_s\Delta\boldsymbol{x}_s^\top - \boldsymbol{\Sigma}_N]$. It then follows from triangle inequality that

$$\|\boldsymbol{E}_s\|_2 \leq \frac{1}{S}\left[\left\|\Delta\boldsymbol{x}_s\Delta\boldsymbol{x}_s^\top\right\|_2 + \|\boldsymbol{\Sigma}_N\|_2\right] = \frac{4R_x}{S}\ . \tag{5.20}$$

As each summand is centered and iid, the variance of sum decomposes as $\mathbb{V}(\Delta_{\boldsymbol{\Sigma}}) = \left\|\sum_{s\in\mathcal{S}}\mathbb{E}[\boldsymbol{E}_s^2]\right\|$. We note that the presence of $\boldsymbol{\Sigma}_N$ in the summands does not hinder independence, as this is just a constant quantity. For a single summand the second moment can be bounded as

$$\begin{aligned}
\mathbb{E}[\boldsymbol{E}_s^2] &= \frac{1}{S^2}\boldsymbol{E}\left[\Delta\boldsymbol{x}_s\Delta\boldsymbol{x}_s^\top - \boldsymbol{\Sigma}_N\right]^2 \\
&= \frac{1}{S^2}\left[\boldsymbol{E}\left[\|\Delta\boldsymbol{x}_s\|_2^2\,\Delta\boldsymbol{x}_s\Delta\boldsymbol{x}_s^\top\right] - \boldsymbol{\Sigma}_N^2\right] \\
&\preceq \frac{1}{S^2}\left[2R_x\boldsymbol{\Sigma}_N - \boldsymbol{\Sigma}_N^2\right] \\
&\preceq \frac{2R_x\boldsymbol{\Sigma}_N}{S^2}\ ,
\end{aligned} \tag{5.21}$$

from which the variance inequality $\mathbb{V}(\Delta_{\boldsymbol{\Sigma}}) \leq \frac{2R_x \|\boldsymbol{\Sigma}_N\|_2}{S}$ follows. Substituting Eqs. (5.20) and (5.21) into Eq. (5.19) yields the result.

(ii) For this part, the scalar version of Eq. (5.19) can be used. We have $\Delta_\sigma = \Delta_{\boldsymbol{w}}^\top(\boldsymbol{\mu}_N - \boldsymbol{\mu}_S) = \sum_{s\in\mathcal{S}} \frac{1}{S}\left[\Delta_{\boldsymbol{w}}^\top(\boldsymbol{\mu}_N - \boldsymbol{x}_s)\right]$ where we denote each scalar summand as $e_s = \frac{1}{S}\left[\Delta_{\boldsymbol{w}}^\top(\boldsymbol{\mu}_N - \boldsymbol{x}_s)\right]$. Once again it is straightforward to verify each summand is centered and iid. An ℓ_1-norm bound can be obtained by Cauchy-Schwarz inequality

$$\begin{aligned} |e_s| &= \left|\frac{1}{S}\Delta_{\boldsymbol{w}}^\top(\boldsymbol{\mu}_N - \boldsymbol{x}_s)\right| \\ &\leq \frac{1}{S}\|\Delta_{\boldsymbol{w}}\|_2 \|\boldsymbol{\mu}_N - \boldsymbol{x}_s\|_2 \\ &= \frac{4\sqrt{R_x R_w}}{S} \,. \end{aligned} \tag{5.22}$$

where for the third line we used the upper bounds $\|\Delta_{\boldsymbol{w}}\|_2 \leq 2\sqrt{R_w}$ and $\|\boldsymbol{\mu}_N - \boldsymbol{x}_s\|_2 \leq 2\sqrt{R_x}$. For the variance we once again have the decomposition $\mathbb{V}(\Delta_\sigma) = \sum_{s\in\mathcal{S}} \mathbb{E}[e_s^2]$ and for a single term we have

$$\begin{aligned} \mathbb{E}[e_s^2] &= \mathbb{E}\left[\left[\frac{1}{S}\Delta_{\boldsymbol{w}}^\top(\boldsymbol{\mu}_N - \boldsymbol{x}_s)\right]^2\right] \\ &= \frac{1}{S^2}\left[\mathbb{E}[\Delta_{\boldsymbol{w}}^\top \boldsymbol{x}_s \boldsymbol{x}_s^\top \Delta_{\boldsymbol{w}}] + \frac{1}{S^2}\mathbb{E}[\Delta_{\boldsymbol{w}}^\top \boldsymbol{\mu}_N \boldsymbol{\mu}_N^\top \Delta_{\boldsymbol{w}}] - \frac{1}{S^2}\mathbb{E}[2\Delta_{\boldsymbol{w}}^\top \boldsymbol{x}_s \boldsymbol{\mu}_N^\top \Delta_{\boldsymbol{w}}]\right] \\ &= \frac{1}{S^2}\Delta_{\boldsymbol{w}}^\top \boldsymbol{\Sigma}_N \Delta_{\boldsymbol{w}} - \frac{1}{S^2}\Delta_{\boldsymbol{w}}^\top \boldsymbol{\mu}_N \boldsymbol{\mu}_N^\top \Delta_{\boldsymbol{w}} \\ &\leq \frac{2R_w \|\boldsymbol{\Sigma}_N\|_2}{S^2} \end{aligned} \tag{5.23}$$

where for the last line we dropped the negative term and used the upper bounds $\|\Delta_{\boldsymbol{w}}\|_2^2 \leq 2R_w$ and $\Delta_{\boldsymbol{w}}^\top \boldsymbol{\Sigma}_N \Delta_{\boldsymbol{w}} \leq \|\Delta_{\boldsymbol{w}}\|_2^2 \|\boldsymbol{\Sigma}_N\|_2^2$. Note that the first upper bound holds by triangle inequality, and the second one follows from the maximum eigenvalue bound. Plugging Eqs. (5.22) and (5.23) into Eq. (5.19) we get the desired result. ■

Both parts of Lemma 1 utilize the unbiased statistics of $\boldsymbol{\mu}_N$ and $\boldsymbol{\Sigma}_N$ and use the scalar and matrix Bernstein inequalities. This is made possible by the unbiased sampling procedure, which is the mean reason we use sampling with repetition. Using Lemma 1 we can now state our main result.

Theorem 2: Let $\boldsymbol{w}_S$ be the solution returned by MBA using S samples. For $\epsilon > 0$, if

$$S \geq \max\left\{ \log(4d/p)\frac{[48R_w^2 \left\|\boldsymbol{\Sigma}_N\right\|_2 + 16\epsilon R_w]R_x}{3\epsilon^2} \ , \ \log(4/p)\frac{48R_w \left\|\boldsymbol{\Sigma}_N\right\|_2 + 16\epsilon\sqrt{R_x R_w}}{3\epsilon^2} \right\} \tag{5.24}$$

then $\left\|\boldsymbol{w}_N - \boldsymbol{w}_S\right\|_2 \leq \delta$ with probability at least $1 - p$.

Proof: The starting point of the proof is the equality

$$\mathcal{L}_S(\boldsymbol{w}_N) - \mathcal{L}_S(\boldsymbol{w}_S) = \mathcal{L}_N(\boldsymbol{w}_N) + \Delta(\boldsymbol{w}_N) - \mathcal{L}_N(\boldsymbol{w}_S) - \Delta(\boldsymbol{w}_S) \ . \tag{5.25}$$

Here by construction

$$\mathcal{L}_N(\boldsymbol{w}_S) - \mathcal{L}_N(\boldsymbol{w}_N) = \mathcal{L}_N(\boldsymbol{w}_S) - \arg\min_{\boldsymbol{w}} \mathcal{L}_N(\boldsymbol{w}) \geq 0 \tag{5.26}$$

$$\mathcal{L}_S(\boldsymbol{w}_N) - \mathcal{L}_S(\boldsymbol{w}_S) = \mathcal{L}_S(\boldsymbol{w}_N) - \arg\min_{\boldsymbol{w}} \mathcal{L}_S(\boldsymbol{w}) \geq 0 \tag{5.27}$$

and we obtain

$$0 \leq \mathcal{L}_N(\boldsymbol{w}_S) - \mathcal{L}_N(\boldsymbol{w}_N) \leq \Delta(\boldsymbol{w}_N) - \Delta(\boldsymbol{w}_S) \ . \tag{5.28}$$

The left hand size of the inequality is a direct consequence of Eq. (5.26). For the right-hand side note that Eq. (5.25) can be manipulated as

$$\begin{aligned} \mathcal{L}_N(\boldsymbol{w}_S) - \mathcal{L}_N(\boldsymbol{w}_N) &= \Delta(\boldsymbol{w}_N) - \Delta(\boldsymbol{w}_S) - [\mathcal{L}_S(\boldsymbol{w}_N) - \mathcal{L}_S(\boldsymbol{w}_S)] \\ &\leq \Delta(\boldsymbol{w}_N) - \Delta(\boldsymbol{w}_S) \end{aligned} \tag{5.29}$$

where the second line follows from $[\mathcal{L}_S(\boldsymbol{w}_N) - \mathcal{L}_S(\boldsymbol{w}_S)] \geq 0$ due to Eq. (5.27). It is therefore sufficient to show that $\Delta(\boldsymbol{w}_N) - \Delta(\boldsymbol{w}_S) \leq \epsilon$ with high probability; the result then follows from the strict convexity argument. Further expand this bounding term as

$$\begin{aligned} \Delta(\boldsymbol{w}_N) - \Delta(\boldsymbol{w}_S) &= \frac{1}{2}\boldsymbol{w}_N^\top(\boldsymbol{\Sigma}_S - \boldsymbol{\Sigma}_N)\boldsymbol{w}_N - \boldsymbol{w}_N(\boldsymbol{\mu}_N - \boldsymbol{\mu}_S) \\ &\quad - \frac{1}{2}\boldsymbol{w}_N^\top(\boldsymbol{\Sigma}_S - \boldsymbol{\Sigma}_N)\boldsymbol{w}_N + \boldsymbol{w}_S^\top(\boldsymbol{\mu}_N - \boldsymbol{\mu}_S) \\ &= \frac{1}{2}\boldsymbol{w}_N{}^\top\Delta_{\boldsymbol{\Sigma}}\boldsymbol{w}_N - \frac{1}{2}\boldsymbol{w}_S{}^\top\Delta_{\boldsymbol{\Sigma}}\boldsymbol{w}_S + (\boldsymbol{w}_S - \boldsymbol{w}_N)^\top(\boldsymbol{\mu}_N - \boldsymbol{\mu}_S) \end{aligned} \tag{5.30}$$

where we used $\Delta_{\boldsymbol{\Sigma}} = \boldsymbol{\Sigma}_S - \boldsymbol{\Sigma}_N$ in the second line. Using triangle inequality we can write

$$|\Delta(\boldsymbol{w}_N) - \Delta(\boldsymbol{w}_S)| \leq \left|\frac{1}{2}\boldsymbol{w}_N{}^\top \Delta_{\boldsymbol{\Sigma}} \boldsymbol{w}_N - \frac{1}{2}\boldsymbol{w}_S{}^\top \Delta_{\boldsymbol{\Sigma}} \boldsymbol{w}_S\right| + \left|(\boldsymbol{w}_S - \boldsymbol{w}_N)^\top (\boldsymbol{\mu}_N - \boldsymbol{\mu}_S)\right| \tag{5.31}$$

and uniformly bound the terms on the right-hand side as:

- Quadratic: $\left|\frac{1}{2}\boldsymbol{w}_N{}^\top \Delta_{\boldsymbol{\Sigma}} \boldsymbol{w}_N - \frac{1}{2}\boldsymbol{w}_S{}^\top \Delta_{\boldsymbol{\Sigma}} \boldsymbol{w}_S\right| < \frac{\epsilon}{2}$
- Linear: $\left|(\boldsymbol{w}_S - \boldsymbol{w}_N)^\top (\boldsymbol{\mu}_N - \boldsymbol{\mu}_S)\right| < \frac{\epsilon}{2}$

For the quadratic term we have

$$\begin{aligned}\left|\frac{1}{2}\boldsymbol{w}_N{}^\top \Delta_{\boldsymbol{\Sigma}} \boldsymbol{w}_N - \frac{1}{2}\boldsymbol{w}_S{}^\top \Delta_{\boldsymbol{\Sigma}} \boldsymbol{w}_S\right| &\leq \frac{1}{2}\left|\boldsymbol{w}_N{}^\top \Delta_{\boldsymbol{\Sigma}} \boldsymbol{w}_N\right| + \frac{1}{2}\left|\boldsymbol{w}_S{}^\top \Delta_{\boldsymbol{\Sigma}} \boldsymbol{w}_S\right| \\ &\leq \frac{1}{2} R_w \left\|\Delta_{\boldsymbol{\Sigma}}\right\|_2 + \frac{1}{2} R_w \left\|\Delta_{\boldsymbol{\Sigma}}\right\|_2 \\ &= R_w \left\|\Delta_{\boldsymbol{\Sigma}}\right\|_2 \ . \end{aligned} \tag{5.32}$$

Here the first line is obtained via triangle inequality; for the second line we use the maximum eigenvalue inequality $\boldsymbol{w}^\top \Delta_{\boldsymbol{\Sigma}} \boldsymbol{w} \leq \|\boldsymbol{w}\|_2^2 \left\|\Delta_{\boldsymbol{\Sigma}}\right\|_2$. We now want to bound $\left\|\Delta_{\boldsymbol{\Sigma}}\right\|_2 \leq \epsilon/(2R_w)$ with probability at least $p/2$. Applying Lemma 2(i) with threshold $\gamma = \epsilon/(2R_w)$ and probability level $p/2$, we obtain the first term in Eq. (5.24).

Secondly, since $\|\boldsymbol{w}_S\|_2^2, \|\boldsymbol{w}_N\|_2^2 \leq R_w$ the the bound for $|\Delta_\sigma|$ in Lemma 2(ii) is directly applicable for the linear term, where we take $\boldsymbol{w}_1 = \boldsymbol{w}_S$ and $\boldsymbol{w}_2 = \boldsymbol{w}_N$. This means, we want to achieve $|\Delta_\sigma| \leq \epsilon/2$ with probability at least $p/2$. Applying Lemma 2(ii) with threshold $\gamma = \epsilon/2$ and probability level $p/2$, we obtain the second term in Eq. (5.24). Since both terms are bounded with probability at least $1 - p/2$, the theorem now follows from the union bound. ■

Theorem 2 shows that the number of samples S required to guarantee $\|\boldsymbol{w}_N - \boldsymbol{w}_S\|_2 \leq \delta$ with high probability does not depend on the total number of pairs $N = N_+ N_-$ provided. Instead the sample size grows logarithmically with the feature size.

This result is useful in that, even though the total number of pairs in the data is too large, randomly sampling a small fraction guarantees a solution that is close to the true solution.

Remark: It might at first seem unreasonable that S does not depend on N; but only on the input dimension d. The reason for this favorable result is that, the given samples are *fixed*, from which uniform subsamples are obtained. So in this context N is actually the support of the discrete distribution on samples. Since the concentration inequalities do not depend on the support size, we do not have N in Eq. (5.24).

While linear models are oftentimes quite competitive, in practice we can have datasets that are not linearly separable; in such cases one is also concerned with devising a nonlinear feature transform. In fact, for finite-dimensional transforms Theorem 2 readily extends. Such transforms include, for example, polynomial features and conjunctions, random Fourier features [93], and random shallow neural networks [94]. In more abstract terms, all these transformations are mappings from d dimensions to F dimensions. Given such fixed transformation, the result of Theorem 2 still holds, where we replace d by F. Therefore, when the input space is not good for a linear ranking function, we can first apply a feature transform, and then the MBA can be used without any modification. This nonlinear version of MBA is summarized in Algorithm 5.2.

5.5 Algorithms and Complexity

5.5.1 Linear Mini-Batch AUC Maximization

Algorithm 5.1 Linear Mini-Batch AUC Maximization

Input: $\boldsymbol{X}^+$, $\boldsymbol{X}^-$ (positive and negative features)

B (mini-batch size), T (rounds), λ_1, λ_2 (regularization)

Output: $\boldsymbol{w}^\star$

Initialize: $\boldsymbol{\mu}_S \leftarrow \mathbf{0}$ and $\boldsymbol{\Sigma}_S \leftarrow \mathbf{0}$.

`// Accumulation`

for $t = 1, \ldots, T$ **do**

Construct index set $\mathcal{S}_t^+$ of size B sampling positive examples uniformly with replacement.

Construct index set $\mathcal{S}_t^-$ of size B sampling negative examples uniformly with replacement.

Construct $\mathcal{S}_t(i) = (\mathcal{S}_t^+(i), \mathcal{S}_t^-(i)),\ i = 1, \ldots, B$.

$\boldsymbol{\mu}_S \leftarrow \boldsymbol{\mu}_S + \frac{1}{BT} \sum_{(i,j)\in\mathcal{S}_t} [\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-]$

$\boldsymbol{\Sigma}_S \leftarrow \boldsymbol{\Sigma}_S + \frac{1}{BT} \sum_{(i,j)\in\mathcal{S}_t} [\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-]\ [\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-]^\top$

end for

`// Global risk minimization`

$\boldsymbol{w}^\star \leftarrow \arg\min_{\boldsymbol{w}} \frac{1}{2} \boldsymbol{w}^\top \boldsymbol{\Sigma}_S \boldsymbol{w}^\top - \boldsymbol{w}^\top \boldsymbol{\mu}_S + \lambda_1 \|\boldsymbol{w}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{w}\|_2^2$

Note: Here $\boldsymbol{X}^+$ and $\boldsymbol{X}^-$ correspond to the $d \times N_+$ and $d \times N_-$ dimensional matrices of positive and negative instances. The indices in the sets $\mathcal{S}$, $\mathcal{S}^+$ and $\mathcal{S}^-$ correspond to column numbers. Consequently, each vector $\boldsymbol{x}_i^+$ or $\boldsymbol{x}_j^-$ is $d \times 1$, $\boldsymbol{\mu}_S$ is $d \times 1$, and $\boldsymbol{\Sigma}_S$ is $d \times d$.

5.5.2 Nonlinear Mini-Batch AUC Maximization

Algorithm 5.2 Nonlinear Mini-Batch AUC Maximization

Input: $\boldsymbol{X}^+$, $\boldsymbol{X}^-$ (positive and negative features)

B (mini-batch size), T (rounds), λ_1, λ_2 (regularization)

$\boldsymbol{\phi}(\cdot)$ (nonlinear feature transform)

Output: $\boldsymbol{w}^\star$

Initialize: $\boldsymbol{\mu}_S \leftarrow \mathbf{0}$ and $\boldsymbol{\Sigma}_S \leftarrow \mathbf{0}$.

`// Accumulation`

for $t = 1, \ldots, T$ **do**

Construct index set $\mathcal{S}_t^+$ of size B sampling positive examples uniformly with replacement.

Construct index set $\mathcal{S}_t^-$ of size B sampling negative examples uniformly with replacement.

Construct $\mathcal{S}_t(i) = (\mathcal{S}_t^+(i), \mathcal{S}_t^-(i)),\ i = 1, \ldots, B.$

$\boldsymbol{\mu}_S \leftarrow \boldsymbol{\mu}_S + \frac{1}{BT} \sum_{(i,j) \in \mathcal{S}_t} [\boldsymbol{\phi}(\boldsymbol{x}_i^+) - \boldsymbol{\phi}(\boldsymbol{x}_j^-)]$

$\boldsymbol{\Sigma}_S \leftarrow \boldsymbol{\Sigma}_S + \frac{1}{BT} \sum_{(i,j) \in \mathcal{S}_t} [\boldsymbol{\phi}(\boldsymbol{x}_i^+) - \boldsymbol{\phi}(\boldsymbol{x}_j^-)]\ [\boldsymbol{\phi}(\boldsymbol{x}_i^+) - \boldsymbol{\phi}(\boldsymbol{x}_j^-)]^\top$

end for

`// Global risk minimization`

$\boldsymbol{w}^\star \leftarrow \arg\min_{\boldsymbol{w}} \frac{1}{2} \boldsymbol{w}^\top \boldsymbol{\Sigma}_S \boldsymbol{w}^\top - \boldsymbol{w}^\top \boldsymbol{\mu}_S + \lambda_1 \|\boldsymbol{w}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{w}\|_2^2$

Note: Here $\boldsymbol{X}^+$ and $\boldsymbol{X}^-$ correspond to the $d \times N_+$ and $d \times N_-$ dimensional matrices of positive and negative instances. The indices in the sets $\mathcal{S}$, $\mathcal{S}^+$ and $\mathcal{S}^-$ correspond to column numbers. We apply the nonlinear feature transform $\boldsymbol{\phi}(\cdot) : \mathbb{R}^d \to \mathbb{R}^F$. Then, each vector $\boldsymbol{\phi}(\boldsymbol{x}_i^+)$ or $\boldsymbol{\phi}(\boldsymbol{x}_j^-)$ is $F \times 1$, $\boldsymbol{\mu}_S$ is $F \times 1$, and $\boldsymbol{\Sigma}_S$ is $F \times F$.

5.5.3 Complexity Analysis

For the MBA algorithm there are two phases: The accumulation phase where the samples are used to compute $\boldsymbol{\mu}_S$ and $\boldsymbol{\Sigma}_S$ and solution phase where we find the optimal weight vector $\boldsymbol{w}^\star$ for the global risk minimization problem of Eq. (5.14). The latter does not depend on the sample size S and it is polynomial in the feature dimensions d. For instance when $\lambda_1 = 0$ the solution can be found by matrix inversion which is $O(d^3)$. Therefore the bottleneck is in the accumulation phase, where the covariance matrix is computed. In particular when the features are dense the total cost of the loop is $O(Sd^2)$ where $S = BT$ is the total number of samples used. Therefore the complexity of MBA is $O(Sd^2)$.

When the nonlinear version of MBA is used we instead get $O(SF^2)$ as the only difference is the feature dimensionality. This is the case when, for example, we use random features approximating kernels, which give F dense features. Therefore the complexity grows quadratically with feature dimension. With that said, the complexity can also can be significantly lower if the input vectors are sparse; however even there the resulting covariance matrix may have too many entries to store in the memory.

Compared to this, the stochastic gradient descent (SGD) methods have an advantage, as the complexity for S samples is $O(Sd)$. Therefore SGD can be chosen when the input dimension prohibitively high. On the other hand, if the input is sparse, dimensionality reduction can first be applied to the inputs, which can then be followed by MBA. An important advantage of MBA is that, there is no learning rate to tune—unlike SGD. This can save significant tuning effort and cross-validation runs.

5.6 Experiments

In this section we conduct four types of experiments to demonstrate the performance benefits of MBA. In the first part we use simulation data from Gaussian mixtures to investigate the performance of various methods; furthermore since we know the data generating distribution, we can also compare with the theoretically optimum Neyman-Pearson decision rule (reviewed in Appendix 5.8.1). Here we show that MBA can achieve better performance with lower number of samples, corroborating the theoretical analysis. For the second part, we experiment with 15 frequently used benchmark datasets from the UCI[4] and LIBSVM[5] repositories; these datasets cover a wide range of application domains and show MBA performs better than the competing methods overall. We then use the same datasets in the third part to demonstrate how the linear framework can be extended to account for nonlinear features. Finally we consider large scale Click Through Rate (CTR) prediction problem with two publicly available commercial datasets with tens of millions of samples. All datasets we use are summarized in Table 5.1. For comparisons we use the following:

1. OLR: Online logistic regression based on SGD.
2. SOLR: Sparse online logistic regression [82].
3. OAM: Online AUC Maximization—the first proposed online AUC maximization algorithm using stochastic gradients [118]. Uses PHL.
4. AdaAUC: Adaptive gradient AUC maximization algorithm in [32]. Proposed as an improvement to the One Pass AUC optimization algorithm in [39]. Uses PSL.
5. MB-PHL: Mini-batch gradient descent algorithm which uses PHL.

[4] `https://archive.ics.uci.edu/ml`

[5] `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

6. MB-PSL: Mini-batch gradient descent algorithm which uses PSL.

7. MBA-ℓ_2: MBA with Tikhonov regularization.

8. MBA-ℓ_2: MBA with Lasso regularization.

9. MBA-EL: MBA with Elastic Net regularization.

MB-PHL and MB-PSL can be thought of a substitutions for OAM and AdaAUC for larger datasets. which we use for the large datasets in this section.

Table 5.1: Summary statistics of datasets used in experiments. For each dataset we show the train/test sample size, feature size, and the ratio of negative samples to positive samples in the training set.

Dataset	# Samp.	# Feat.	T_- / T_+
a1a	1.6K / 30.9K	123	3.06
a9a	32.5K / 16.2K	123	3.15
amazon	750 / 750	10,000	2.33
bank	20.6K / 20.6K	100	7.88
codrna	29.8K / 29.8K	8	2.00
german	500 / 500	24	2.33
ijcnn	50K / 92K	22	9.30
madelon	2,000 / 600	500	1.00
mnist	60K / 10K	780	2.30
mushrooms	4K / 4K	112	0.93
phishing	5.5K / 5.5K	68	0.79
svmguide3	642 / 642	21	2.80
usps	7.2K / 2K	256	2.61
w1a	2.5K / 47.2K	300	33.40
w7a	25K / 25K	300	32.40
avazu app	12.6M / 2M	10,000	8.33
avazu site	23.6M / 2.6M	10,000	4.06
criteo	45.8M / 6M	10,000	2.92

5.6.1 Simulation Study

For the simulations we consider the binary hypothesis testing problem of Eq. (5.35), which can represent, for example, a radar or telecommunication setting. In particular, we employ Gaussian mixtures as data generating distributions. Namely, for hypothesis-i a K-component Gaussian mixture is given by

$$p_i(\boldsymbol{x}) = \sum_{k=1}^{K} c_{ik}\ (2\pi)^{-d/2}\ |\boldsymbol{\Sigma}_{ik}|^{-1/2} \times \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_{ik})^\top \boldsymbol{\Sigma}_{ik}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_{ik})\right\}\ , \tag{5.33}$$

where the weights c_{ik} are coefficients on the probability simplex. This distribution is completely characterized by the weights, means, and covariances. For hypothesis-i let c_i, $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ denote the *set* of these parameters. For our experiments $K \in \{1, 2, 3\}$ and:

- $k = 1$: We set $c_0 = \{1\}$, $\boldsymbol{\mu}_0 = \{-\mathbf{0.1}\}$, $\boldsymbol{\Sigma}_0 = \{\boldsymbol{I}\}$ and $c_1 = \{1\}$, $\boldsymbol{\mu}_1 = \{\mathbf{0.1}\}$, $\boldsymbol{\Sigma}_1 = \{\boldsymbol{I}\}$.
- $k = 2$: We set $c_0 = \{0.9, 0.1\}$, $\boldsymbol{\mu}_0 = \{-\mathbf{0.1}, \mathbf{0.1}\}$, $\boldsymbol{\Sigma}_0 = \{\boldsymbol{I}, \boldsymbol{I}\}$ and $c_1 = \{0.1, 0.9\}$, $\boldsymbol{\mu}_1 = \{-\mathbf{0.1}, \mathbf{0.1}\}$, $\boldsymbol{\Sigma}_1 = \{\boldsymbol{I}, \boldsymbol{I}\}$.
- $k = 3$: We set $c_0 = \{0.8, 0.1, 0.1\}$, $\boldsymbol{\mu}_0 = \{-\mathbf{0.1}, \mathbf{0}, \mathbf{0.1}\}$, $\boldsymbol{\Sigma}_0 = \{\boldsymbol{I}, \boldsymbol{I}, \boldsymbol{I}\}$ and $c_1 = \{0.1, 0.1, 0.8\}$, $\boldsymbol{\mu}_1 = \{-\mathbf{0.1}, \mathbf{0}, \mathbf{0.1}\}$, $\boldsymbol{\Sigma}_1 = \{\boldsymbol{I}, \boldsymbol{I}, \boldsymbol{I}\}$

As it can be seen the distributions we choose for the two hypotheses are symmetric across the origin. As the number of components increase the distributions get less interspersed and the problem becomes more challenging. All covariances are set to identity; we finally note that the bold numbers for the mean sets correspond to a vector of the bold entry replicated. For our experiments, for each value of K we form 50 training sets, where for each dataset we sample 20,000 points from the generative distribution. Furthermore we create an imbalanced dataset, where roughly 90% of

the data has label 0 (i.e. sampled from hypothesis 0). We also create a separate test set, with 100,000 samples and the same imbalance.

Since the generative distributions are assumed known in this setting, we can analytically derive the Neyman-Pearson (NP) decision rule which maximizes the AUC. For any given K, the NP rule computes the scores through $p_1(\boldsymbol{x})/p_0(\boldsymbol{x})$. Note that, this does not require any training data as the generating distributions are already known. A particularly interesting case is $K = 1$. Here the log likelihood is [6]

$$\log \frac{p_1(\boldsymbol{x})}{p_0(\boldsymbol{x})} = \boldsymbol{x}^\top(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) + \frac{1}{2}(\boldsymbol{\mu}_0^\top \boldsymbol{\mu}_0 - \boldsymbol{\mu}_1^\top \boldsymbol{\mu}_1) \tag{5.34}$$

Since the constant term does not affect AUC, we see that the optimum ranking rule is a linear function of $\boldsymbol{x}$. So for this specific case, the class of linear discriminants (i.e. $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$) is consistent [87].

For the comparisons in this section we use MBA-ℓ_2, ONLR, and AdaAUC, as the latter two typically have the best competitive performance against the former. We run the experiments for three different sample ratio (SR), namely SR $\in \{1\%, 10\%, 100\%\}$; this represents the percentage of available data points used for training. As mentioned above we average the generalization performance over 50 training samples, and report the average values and standard deviations. Table 5.2 shows the resulting AUC values and Figure 5.1 displays the corresponding ROC curves.

Firstly note that, as K increases, the AUC value achieved by the optimal NP rule decreases; this shows that adding more mixture components progressively makes the problem harder. As we increase the SR, all three learning algorithms improve, as expected. However, we can see that the starting point for MBA is significantly higher than the other two. In particular, AdaAUC is worse for small sample sizes. This shows the difficulty of optimizing a bivariate loss function as opposed to the univariate logistic loss of ONLR. The particular difficulty comes from selecting the step size, and for smaller number of samples the stochastic gradient is inefficient.

[6] In fact the linearity holds for arbitrary $\boldsymbol{\Sigma}$ as long as it is shared by both hypotheses.

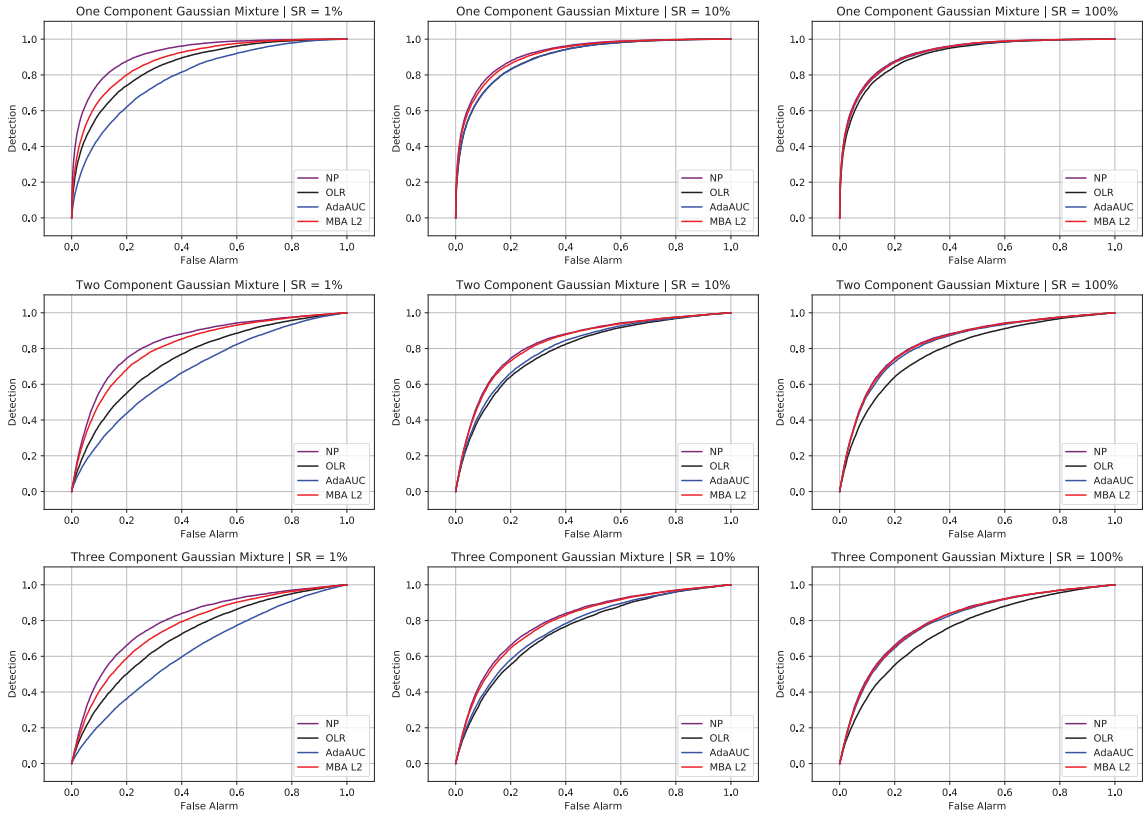


Figure 5.1: The ROC curves obtained by the Neyman-Pearson detector and three learning algorithms on the simulated data. The rows are in increasing order of mixture components (K) and the columns are in increasing order of sample ratio (SR).

Being learning rate and gradient free, MBA does not suffer from these drawbacks and always performs better than ONLR.

We also use pairwise t-test to assess the statistical significance of results, using 95% confidence level, as proposed and used by [39] initially for this problem. For all cases considered we see that MBA achieves better results than its competitors with significance. This is true even when $SR = 100\%$ and the average values are close, as the standard deviations are low and a high number of experiments are performed.

Interestingly, comparing the performance of NP and MBA we see that in all three

Table 5.2: Comparisons of algorithms on simulated data The performance of MBA-ℓ_2, ONLR, and AdaAUC are reported for $k \in \{1, 2, 3\}$ and SR $\in \{1\%, 10\%, 100\%\}$. The symbols filled/empty circle indicate that MBA is (statistically) significantly better/worse.

Distribution	SR	Neyman-Pearson	MBA-ℓ_2	ONLR	AdaAUC
1-Component	1 %		87.43 ± 0.69	• 86.79 ± 0.99	• 80.94 ± 2.77
Gaussian	10 %	92.13	91.44 ± 0.10	• 90.54 ± 0.29	• 90.25 ± 0.31
Mixture	100 %		91.88 ± 0.04	• 90.69 ± 0.24	• 91.70 ± 0.07
2-Component	1 %		80.15 ± 0.68	• 77.64 ± 1.41	• 69.75 ± 3.24
Gaussian	10 %	83.71	83.15 ± 0.10	• 80.19 ± 0.69	• 80.12 ± 0.69
Mixture	100 %		83.47 ± 0.04	• 80.19 ± 0.69	• 83.01 ± 0.11
3-Component	1 %		76.39 ± 0.47	• 73.07 ± 1.06	• 66.38 ± 1.95
Gaussian	10 %	80.22	79.52 ± 0.11	• 75.94 ± 0.64	• 76.72 ± 0.33
Mixture	100 %		79.93 ± 0.11	• 75.91 ± 0.83	• 79.61 ± 0.06

cases the achieved AUC is quite close. This is the case even when $K > 1$; therefore even if the optimal scores are a nonlinear function of $\boldsymbol{x}$ for these cases, the linear approximation is still reasonable. With that said, if the data was highly nonlinear, a linear ranking function would not be effective.

5.6.2 UCI and LIBSVM Benchmark Data

In this section we experiment with 15 benchmark datasets from the UCI and LIBSVM repositories which we summarize in Table 5.1. It can be seen that the chosen datasets cover a wide range of sample/feature sizes. The distribution of samples vary from being linearly separable to highly nonlinear. The datasets also exhibit significant differences in label imbalance. In terms of the features present, datasets fall into

three categories: numerical only, categorical only, and mixed. We use the train/test splits provided in LIBSVM website; if splits are not available we use 50/50 splitting with stratification. For multiclass datasets we map the classes to binary labels.

Table 5.3 shows the AUC values obtained by six competing algorithms on benchmark datasets. Here the results are reported along with standard deviations. In addition, we once again conduct a pairwise t-test with 95% significance level. To perform this test, we compare each algorithm in the last four columns to the two MBA algorithms in the first two columns. If MBA performs significantly better/worse we represent this with a filled/empty circle. Table 5.3 shows that there is a clear benefit in using the proposed MBA, whereas the recent AdaAUC is the second best competitor. The mini-batch processing phase of MBA is learning-rate free and this brings an important advantage. AdaAUC adapts the gradient steps, while we used the learning rate $O(1/\sqrt{t})$ for all other stochastic gradient algorithms, which performed well with this choice. However, though MBA does not need this parameter, it still performs significantly better than AdaAUC in 9/15 cases. It is also worth noting that MBA-ℓ_1 obtains 100% AUC for the `mushrooms` data, and for the `svmguide3` dataset MBA is at least 9% better than the others. While logistic regression does not directly optimize AUC, it is frequently used in practice, where AUC is the main metric, as it typically has competitive performance. Here we see that logistic regression has a decent performance as well, and in fact beats MBA on the `a9a` data.

Another important performance measure is the ability to rank as a function of sample size. We show comparisons for this in Figure 5.2. We see that the stochastic gradient-based methods keep improving as the sample size increases, whereas MBA has a relatively steady performance, and it converges faster. This indicates that, for these benchmark datasets MBA can already construct a good approximation of the global problem at this point. This result is not surprising given Theorem 4, as good performance is independent of the number of pairs or instances, and only related to the dimensionality of the optimization problem to be solved. In the first panel of

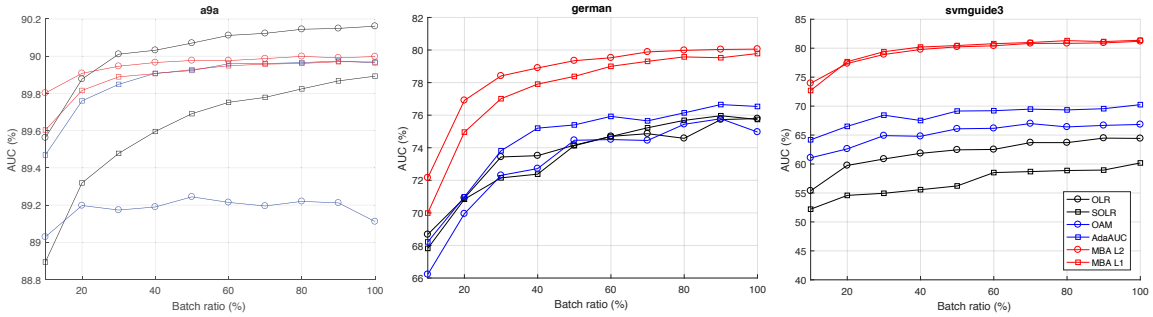


Figure 5.2: AUC performance of six algorithms as a function of sample size for a9a, german, and svmguide3 selected from LIBSVM.

Figure 5.2 the best performer is logistic regression although the difference is rather small. In the second plot, MBA-ℓ_2 gives the best result, although AdaAUC is good as well. For the other two plots, both MBA methods have a clear advantage from start to finish.

Table 5.3: Comparison of algorithms on 15 benchmark datasets from UCI and LIBSVM repositories. The symbols filled/empty circle indicate one of the MBA is (statistically) significantly better/worse.

Dataset	MBA-ℓ_2	MBA-ℓ_1	OLR	SOLR	OAM	AdaAUC
a1a	88.98 ± 0.14	88.67 ± 0.15	• 88.64 ± 0.27	• 88.04 ± 0.14	• 87.61 ± 0.45	• 88.51 ± 0.34
a9a	89.97 ± 0.01	89.97 ± 0.02	◦ 90.17 ± 0.03	• 89.88 ± 0.03	• 89.30 ± 0.22	89.99 ± 0.04
amazon	77.12 ± 0.44	71.35 ± 2.50	• 69.90 ± 2.21	• 71.87 ± 0.74	• 60.23 ± 3.90	• 74.97 ± 0.89
bank	93.22 ± 0.06	93.22 ± 0.03	• 82.89 ± 0.21	• 80.23 ± 0.33	• 81.51 ± 0.49	• 89.46 ± 0.12
codrna	97.68 ± 0.00	97.63 ± 0.01	• 95.69 ± 0.19	• 92.36 ± 0.85	• 97.31 ± 0.12	• 94.34 ± 0.40
german	80.34 ± 0.80	80.41 ± 0.64	• 76.39 ± 1.69	• 75.07 ± 1.22	• 74.59 ± 1.79	• 77.83 ± 1.29
ijcnn	90.53 ± 0.05	90.40 ± 0.07	• 89.50 ± 0.53	• 88.93 ± 0.48	• 88.52 ± 1.76	90.59 ± 0.28
madelon	62.39 ± 0.44	62.34 ± 0.51	61.97 ± 0.69	• 61.81 ± 0.48	• 60.64 ± 0.44	61.82 ± 1.58
mnist	95.81 ± 0.02	95.77 ± 0.02	• 95.63 ± 0.27	• 95.49 ± 0.12	• 94.82 ± 0.23	• 95.47 ± 0.09
mushrooms	100.00 ± 0.00	100.00 ± 0.00	99.88 ± 0.03	• 99.73 ± 0.07	• 99.62 ± 0.28	• 99.98 ± 0.00
phishing	98.32 ± 0.01	98.32 ± 0.05	98.49 ± 0.01	98.38 ± 0.03	• 98.08 ± 0.27	98.36 ± 0.02
svmguide3	81.16 ± 0.80	82.05 ± 0.66	• 63.80 ± 0.81	• 57.65 ± 2.98	• 66.97 ± 3.45	• 69.14 ± 1.95
usps	95.89 ± 0.04	95.83 ± 0.06	• 95.82 ± 0.15	• 95.71 ± 0.07	• 94.65 ± 0.53	• 95.74 ± 0.13
w1a	92.28 ± 0.23	91.21 ± 0.36	• 84.81 ± 1.34	• 79.84 ± 1.15	• 87.83 ± 1.59	• 90.70 ± 0.67
w7a	96.27 ± 0.07	96.17 ± 0.08	• 93.05 ± 0.29	• 89.27 ± 0.82	• 93.92 ± 0.55	• 95.09 ± 0.26
Win/Tie/Loss	-	-	11/3/1	14/1/0	15/0/0	11/4/0

5.6.3 Nonlinear Features

So far we have demonstrated how MBA can achieve better performance than its competitors; however we focused on learning a linear ranking function. As mentioned in Section 5.3, MBA can easily be extended to nonlinear feature spaces; in this case we first perform the feature transformation and apply the algorithm to this new set of features. Here we show an application using three datasets from the previous section: `german`, `mnist`, and `svmguide3`. Our baseline will be the best values obtained by the linear MBA (LIN) from Table 5.3. We obtain nonlinear features using a variety of methods: random thresholds to obtain binary features (RT) [94], random Fourier features (RFF) [93] which approximate kernel machines, random neural networks (RNN) where the weights are obtained via randomization, and finally a single layer binary classification network (NN) which is learned from the training data. Note that in the latter case we can also use the network itself to get scores, however we instead feed the representations in the hidden layer to MBA, for demonstration purposes.

For the `german` and `svmguide` datasets we use 500 random features, whereas `mnist` uses 1000 as it has higher dimensionality. For the NN the hidden layer number is half of the input dimension, and both RNN and NN use hyperbolic tangent activation function. NN is trained using Adam optimizer [68]. Given the nonlinear features all the ranking functions are learned with MBA-L2.

We report the test AUC values in Figure 5.3. For the `german` dataset we see that the linear model has the best performance, and applying a nonlinear feature transform actually degrades performance. This is a common theme in learning nonlinear features; in general there is no principled way of selecting the right feature transform, and in our case the approaches we consider fail to find a better representation. In fact, for many datasets it may be difficult to find a nonlinearity that would perform better, and consequently linear models remain highly practical. For the `mnist` dataset, we see that while RT does not provide an improvement, RFF, RNN, and NN all yield substantially higher AUC. This dataset is a widely used benchmark for training neural

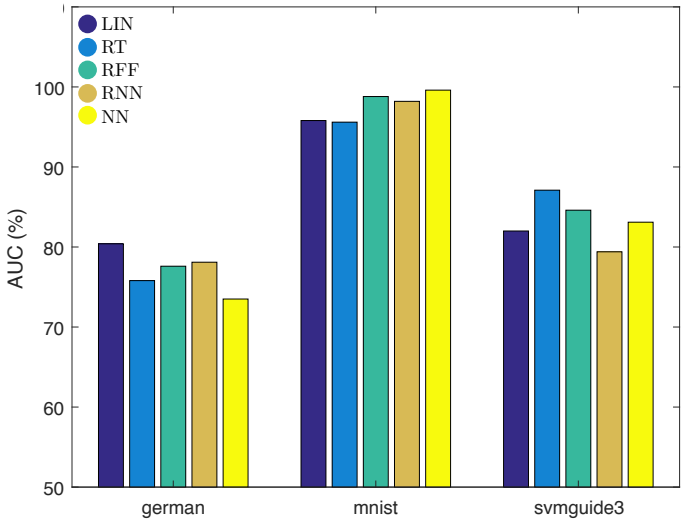


Figure 5.3: AUC performance of four nonlinear feature generation methods, compared to the linear case. All of the training is done via MBA-L2.

networks [105] and the features learned by neural nets are highly effective. The best performance in this case is given by NN, which shows that learning representations in a data-dependent manner is the best choice here. Finally, for the `svmguide3` dataset we see that the best performance is given by RT. While RFF and NN also provide improvement, RT has a clear edge here. This reinforces our previous claim that there is not a one-shot solution to the feature engineering problem and it is rather a process of trial and error.

5.6.4 Large-scale Web Click Data

For this last part of experiments we use large scale datasets where the task is Click Through Rate (CTR) prediction. Estimating user clicks in web advertising is one of the premier areas of AUC optimization. As the number of users who click a given ad is typically low, the task naturally manifests itself as distinguishing click from non-click. The datasets used come from Avazu and Criteo, available at LIBSVM—

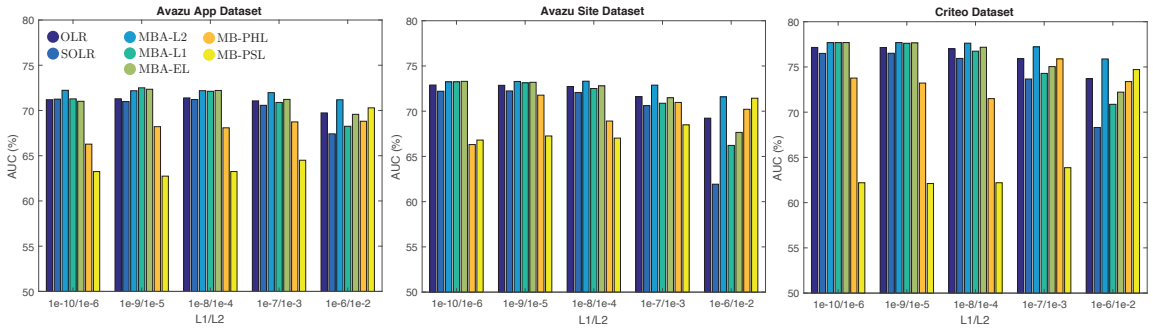


Figure 5.4: AUC achieved by all algorithms on the Avazu App, Avazu Site, and Criteo datasets. Here the performance is plotted as a function of regularization parameters. The elastic net uses one half of ℓ_1-penalty for both ℓ_1 and ℓ_2 regularization.

summarized at the end of Table 5.1. It is worthwhile to note that these datasets are an order of magnitude larger than the ones used in previous studies, showing the scaling benefits of MBA. This time we shuffle and split the entire dataset into chunks of 100 (Avazu App) and 200 (Avazu Site and Criteo). We then make a single pass over these chunks with randomized sampling and report the results. For these datasets the variation across different runs is very small, as the inputs are very uniform. Therefore we do not show the confidence intervals in the bar charts, but note that all results are statistically significant. For the CTR problem, a 0.1% improvement in AUC is considered significant, whereas an increase of 0.5% results in noticable revenue gain.

In Figure 5.4 we show the AUC performance of seven algorithms. For the Avazu App data, MBA-ℓ_2 gives the best results while for Avazu Site and Criteo all MBA algorithms give similar results. Comparing the proposed MBA with the best non-MBA algorithm, the performance improvements are 1.20%, 0.43% and 0.54%. The mini-batch gradient descent algorithms do not perform that well, especially when the regularization parameter is small, and they get better as this parameter increases. For these experiments the step size is $\mathcal{O}(1/\sqrt{t})$, and while logistic regression has good

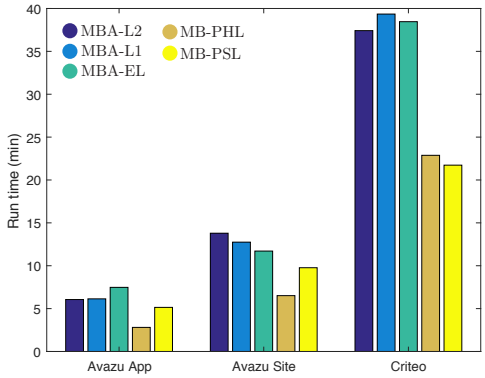


Figure 5.5: Runtime comparison of MBA with MB-PSL and MB-PHL. As the latter two only require a gradient computation they are faster than MBA, but with significantly reduced performance. On the other hand, MBA can process tens of millions of samples under an hour, showing the scalability of this approach.

performance with this choice, optimizing pairwise losses seems less robust. As the regularization increases the variation in the gradients decreases, which helps improve the AUC scores. We also experimented with a small constant step size, which yielded similar results. On the other hand MBA does not require this parameter, making it a better choice.

Another important concern is the running time. Here, we do not make a relative comparison, instead we state how much time it takes to find the result. This is because, comparing the running time to logistic regression is not very informative; if a sequential logistic regression is implemented in Python script, then the mini-batch algorithm is roughly 10 times faster, as sequential processing is slow. However, if an optimized package is used, then it can be 100 times faster than MBA, as the underlying code is optimized. For this reason we show the running time of the vanilla implementation of MBA in Figure 5.5. As it can be seen, even for the Criteo dataset, which contains the largest number of instances, the runtime is under an hour. As we

briefly mentioned in Section 5.3, the mini-batch portion of MBA can be distributed without loss of accuracy, therefore using cluster computing, MBA can easily scale to billion-sample datasets, which are several orders of magnitude larger than the datasets that can be handled by sequential methods.

5.7 Conclusion

In this chapter we have introduced a fast algorithm to maximize the AUC metric. Our proposed approach, called MBA, uses the specific structure of the squared pairwise surrogate loss function. In particular, it is shown that one can approximate the global risk minimization problem simply by approximating the first and second moments of pairwise differences of positive and negative inputs. This suggests an efficient mini-batch scheme, where the moments are estimated by U-statistics. MBA comes with theoretical guarantees, and importantly the number of samples required for good performance is independent of the number of pairs present, which is typically a very large number. Our experiments demonstrate the advantages of MBA in terms of speed and performance. MBA would be particularly useful for applications where AUC is the prime metric, and the data size is massive and parallel processing is necessary.

5.8 Appendix to Chapter 5

5.8.1 AUC Maximization in Signal Detection

In this section we discuss the connection of AUC maximization to the signal detection framework. This framework is concerned with a probabilistic setup, where the optimal solution with maximum AUC can be obtained in analytical form. This is in contrast to the statistical learning setup which assumes that the probability distributions generating the observations are unknown to the modeler.

In binary signal detection we have two hypotheses

$$\mathcal{H}^+ \ : \ \boldsymbol{X} \sim \mathcal{P}^+ \quad , \quad \mathcal{H}^- \ : \ \boldsymbol{X} \sim \mathcal{P}^- \tag{5.35}$$

This setting arises frequently in many different applications, such as radar systems and communication channels [92]. In this setup, it is commonly assumed that the generating distributions $\mathcal{P}^+$ and $\mathcal{P}^-$ are known (c.f. Eq. (5.1)). For our purposes we consider the Neyman-Pearson (NP) hypothesis testing scenario, where neither the priors for hypotheses are known, nor the costs of making a wrong decision. In this case the optimal detector is designed based on the following two metrics: detection and false alarm, defined for a fixed decision rule as

$$\begin{aligned} \text{Detection: } P_D(\Gamma) &= \int \Gamma(\boldsymbol{x}) \ p^+(\boldsymbol{x}) \ d\boldsymbol{x} \\ \text{False Alarm: } P_F(\Gamma) &= \int \Gamma(\boldsymbol{x}) \ p^-(\boldsymbol{x}) \ d\boldsymbol{x} \end{aligned} \tag{5.36}$$

Two immediate observations follow: (i) The metrics measure the performance of the rule itself, so they are a function of Γ. (ii) The detector $\Gamma(\boldsymbol{x})$, in turn, is a mapping from the observed signal $\boldsymbol{x}$ to the hypotheses. As the names imply, detection is the probability of correctly choosing the positive hypothesis, whereas false alarm is incorrectly doing so. In general, the positive hypothesis corresponds to the presence of a target/message, whereas the negative one indicates absence, hence the names.

In NP hypothesis testing, the optimal detector is the solution to the optimization

$$\Gamma'(\boldsymbol{x}) := \arg\max_{\Gamma} P_D(\Gamma) \quad \text{s.t.} \quad P_F(\Gamma) \leq \alpha \ . \tag{5.37}$$

We therefore seek the detector with highest detection probability while setting a limit on the false alarm rate ($0 \leq \alpha \leq 1$). Note that, without this limit (i.e. $\alpha = 1$) we can use a trivial decision rule that maps all observations to positive hypothesis and obtain $P_D(\Gamma) = 1$. The solution is given by the following.

Lemma 1 (Neyman-Pearson, [92]): For α, let Γ be any decision rule with

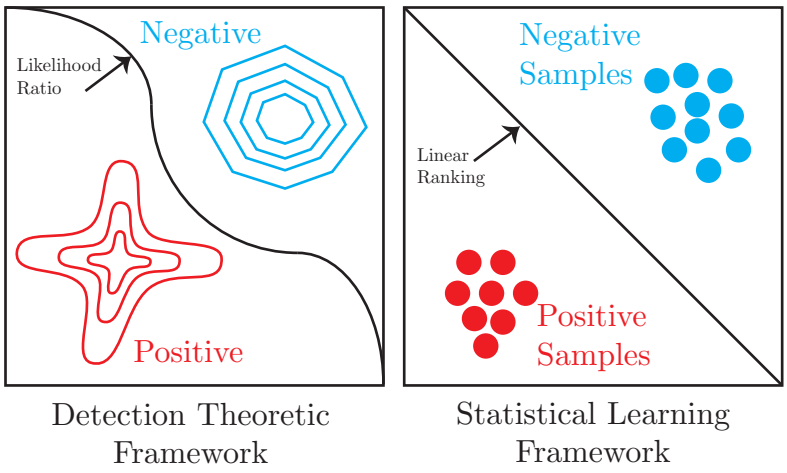


Figure 5.6: A cartoon illustration of the signal detection (left) and statistical learning (right) frameworks for AUC maximization.

$P_D(\Gamma) \leq \alpha$ and let Γ' be the decision rule of form

$$\Gamma'(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad p_1(\boldsymbol{x}) > \eta \; p_0(\boldsymbol{x}) \\ \gamma(x) & \text{if} \quad p_1(\boldsymbol{x}) = \eta \; p_0(\boldsymbol{x}) \\ 0 & \text{if} \quad p_1(\boldsymbol{x}) < \eta \; p_0(\boldsymbol{x}) \end{cases} \tag{5.38}$$

where $\eta \geq 0$ and $0 \leq \gamma(\boldsymbol{x}) \leq 1$ are chosen such that $P_F(\Gamma') = \alpha$. Then $P_D(\Gamma') \geq P_D(\Gamma)$.

We note that a decision rule that is optimal in the NP sense satisfies the false alarm inequality on the boundary. The structure in Eq. (5.38) reveals that, for any given input $\boldsymbol{x}$ this rule computes a score based on the likelihood ratio $p_1(x)/p_0(x)$ and compares it to a threshold. Since the ROC curve is the plot of detection vs false alarm, when $\mathcal{P}^+$ and $\mathcal{P}^-$ are known, the likelihood ratio function can be used to obtain scores with maximum AUC.

The framework outlined in this section is displayed in Figure 5.6, left panel. In general, the likelihood ratio test would yield non-linear decision boundaries. On the

other hand, the right panel of Figure 5.6 displays the statistical learning approach to AUC optimization, where the ϕ-risk is minimized under the linear classifier assumption, i.e. this is what the MBA does. In contrast to the signal detection problem of left panel, here we only have access to samples, instead of the class-conditional densities. As the figure suggests, the linear scoring function assumption is meaningful when the two classes are linearly separable.

Chapter 6

Conclusion

In this thesis I have investigated the problem of dynamic machine learning. Dynamic machine learning is concerned with learning from data, where the data exhibits dependence on time. Such data naturally arises in most settings such as stock prices or item purchases. However, the definition is broad and virtually contains all data, as data collection is done over time. On the other hand, increasing data size and streaming data motivates online learning algorithms. In the online setting, the input is processed as a stream, in a single pass. Once again, there are many applications of this, such as real time prediction for target tracking or financial forecasting.

In Part I of this thesis we considered dynamic matrix factorization. This extends the widely adopted matrix factorization methodology to time varying data. In Section 2 we considered dyadic data, where each observation is a result of an interaction between two entities. When the observation matrix is sparse, low rank matrix factorization is an effective method for handling missing data. Furthermore, the factors can be assigned Brownian motion priors; coupled with Gaussian likelihood functions, the posterior distribution can be approximated by mean-field variational inference, where the posterior distributions are conjugate to the priors. This leads to an efficient online learning algorithm that can learn dynamically changing factors. In Section 3, the dynamic matrix factorization approach is extended to forecasting future values of

high dimensional time series with missing observations. Here the dynamic model is enhanced by using a vector autoregressive model for the low rank time series factor. The coefficients of this model can then be learned in closed form, and recursively, using minimum mean square error estimator. This model notably preserves the forecasting accuracy, even when the percentage of missing values is very high.

In Part II we consider the problem of nonlinear Kalman filtering. The problem arises frequently, when the state-space equations contain nonlinear terms. The model of Section 2—Collaborative Kalman Filter—is an instance where the likelihood term contains an inner product between two hidden state vectors. While in that case variational inference gives closed form updates, for the nonlinear Kalman filtering problem this is not the case in general. We tackle the problem from the lens of divergence minimization; in particular we consider the forward and backward Kullback-Leibler divergences and the alpha divergence. The forward KL divergence is linked to variational inference, which is leveraged to make connections to well established filtering algorithms such as the Extended Kalman Filter. The reverse KL divergence, on the other hand, is linked to expectation propagation, where the posterior is simply obtained by moment matching procedure. Finally, the alpha divergence minimization approach gives a generalized moment matching procedure. The resulting filters improve significantly upon their competitors in literature, and the alpha divergence filter is quite robust to measurement noise and model uncertainties.

In Part III the problem setting is changed to batch, where the aim is to learn ranking from binary labeled data, also known as the bipartite ranking problem. The problem frequently arises in imbalanced and cost-sensitive learning problems, where learning to correctly label is crucial. Interestingly, the problem is rooted in signal detection theory, where for known generating distributions the Neyman-Pearson detector is optimal. Since in many practical cases we cannot compute this detector, a learning approach is taken. In particular we considered a linear model with pairwise squared loss function, where we proved that our algorithm can return a good solu-

tion with high probability, using a very low fraction of available samples. This is an important result, as for this problem the sample size is quadratic in the number of positive and negative instances. Experiments show equal or better area under curve values compared to state-of-the-art methods on various datasets. It also scales well for very large datasets.

As mentioned in the introduction, throughout the chapters, a unifying theme was the least squares loss function as the objective. Taking a closer look, we note that the least square cost manifests itself in the likelihood terms: In Part I this is for the sparse matrix or vector observation, in Part II for nonlinear measurement equation, and in Part III for observed pairs. The least squares loss brings many benefits; in Part I it allows for closed form optimization, in Part II it yields Gaussian approximate posteriors which can be computed efficiently, and in Part III it is leveraged to obtain a learning-rate free, distributed, and asynchronous algorithm.

The ideas explored in this thesis can lead to a multitude of avenues for future research. As the datasets grow in size and the demand for real-time solutions increases, dynamic machine learning will remain a useful tool. The dynamic matrix factorization models of Part I can be applied to other types of data, such as speech, and e-commerce records. The application of dynamic matrix factorization for time series forecasting is a promising are, and extensions to our linear predictor in Section 3 are possible. For Part II, an important extension to divergence minimization-based nonlinear Kalman filtering would be to incorporate multimodal densities which frequently arise in applications; such densities are also useful for nonparametric inference. Finally for Part III, nonlinear extensions to the proposed AUC maximization framework are promising, and could be an alternative to some of the widely used nonlinear prediction models which do not scale well with data size.

Bibliography

[1] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, February 1998.

[2] Oren Anava, Elad Hazan, Shie Mannor, and Ohad Shamir. Online learning for time series prediction. In *Conference on Learning Theory*, 2013.

[3] Oren Anava, Elad Hazan, and Assaf Zeevi. Online time series prediction with missing data. In *International Conference on Machine Learning*, 2015.

[4] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

[5] Aleksandr Y. Aravkin, James V. Burke, and Gianluigi Pillonetto. Optimization viewpoint on kalman smoothing with applications to robust and sparse estimation. In *Compressed Sensing & Sparse Filtering*. Springer, 2014.

[6] Aleksandr Y. Aravkin, Kush R. Varshney, and Dmitry M. Malioutov. A robust nonlinear kalman smoothing approach for dynamic matrix factorization. Technical report, IBM Thomas J. Watson Research Center, 2015.

[7] Aleksandr Y. Aravkin, Kush R. Varshney, and Liu Yang. Dynamic matrix factorization with social influence. In *IEEE International Workshop on Machine Learning for Signal Processing*, 2016.

[8] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, Feb 2002.

[9] Peter L. Bartlett, Michael I. Jordan, and Jon D. McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 2006.

[10] Suat Bayram, San Gultekin, and Sinan Gezici. Noise enhanced hypothesis testing according to restricted neyman pearson criterion. *Digital Signal Processing*, 2014.

[11] Matthew J. Beal. *Variational Algorithms for Approximate Bayesian Inference*. PhD thesis, University of London, 2003.

[12] David Belanger and Sham Kakade. A linear dynamical system model for text. In *International Conference on Machine Learning*, 2015.

[13] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.

[14] David M. Blei and John D. Lafferty. Dynamic topic models. In *International Conference on Machine Learning*, 2006.

[15] Azzedine Boukerche, Horacio A.B. Oliveira, Eduardo F. Nakamura, and Antonio A.F. Loureiro. Localization systems for wireless sensor networks. *IEEE Wireless Communications*, 2007.

[16] Stephen Boyd and Lieven Vanderberghe. *Convex Optimization*. Cambridge University Press, 2004.

[17] Ulf Brefeld and Tobias Scheffer. Auc maximizing support vector learning. In *ICML Workshop on ROC Analysis in Machine Learning*, 2005.

[18] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer, 2016.

[19] Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C. Wilson, and Michael Jordan. Streaming variational bayes. In *Advances in Neural Information Processing Systems*, 2013.

[20] Emmanuel J. Candes and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 2009.

[21] Emmanuel J. Candes and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 2010.

[22] Cristiano L. Castro and Antonio P. Braga. Novel cost-sensitive approach to improve the multilayer perceptron performance on imbalanced data. *IEEE Transactions on Neural Networks and Learning Systems*, 2013.

[23] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[24] Hao Chen, Pramod K. Varshney, Steven M. Kay, and James H. Michels. Theory of the stochastic resonance effect in signal detection: Part I - Fixed detectors. *IEEE Transactions on Signal Processing*, 2007.

[25] Hao Chen, Pramod K. Varshney, Steven M. Kay, and James H. Michels. Theory of the stochastic resonance effect in signal detection: Part II - Variable detectors. *IEEE Transactions on Signal Processing*, 2008.

[26] Jianshu Chen, Zaid J. Towfic, and Ali H. Sayed. Dictionary learning over distributed models. *IEEE Transactions on Signal Processing*, 2015.

[27] Chee Y. Chong and Srikanta P. Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proceedings of the IEEE*, 2003.

[28] Miew K. Choong, Maurice Charbit, and Hong Yan. Autoregressive-model-based missing value estimation for dna microarray time series data. *IEEE Transactions on Information Technology in Biomedicine*, 2009.

[29] Erhan Cinlar. *Probability and Stochastics*. Springer, 2011.

[30] Stephan Clemencon, Gabor Lugosi, and Nicolas Vayatis. Ranking and empirical minimization of u-statistics. *The Annals of Statistics*, 2008.

[31] Yi Ding, Chenghao Liu, Peilin Zhao, and Steven C.H. Hoi. Large scale kernel methods for online auc maximization. In *International Conference on Data Mining (ICDM)*, 2017.

[32] Yi Ding, Peilin Zhao, Steven C. H. Hoi, and Yew Soon Ong. An adaptive gradient method for online auc maximization. In *Association for Advancement of Artificial Intelligence (AAAI)*, 2015.

[33] William Dunsmuir and Peter R. Robinson. Estimation of time series models in the presence of missing data. *Journal of the American Statistical Association*, 1981.

[34] Geir Evensen. The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics*, 2003.

[35] Moein Falahatgar, Mesrob I. Ohannessian, and Alon Orlitsky. Near-optimal smoothing of structured conditional probability matrices. In *Advances in Neural Information Processing Systems*, 2016.

[36] Jiashi Feng, Huan Xu, and Shuicheng Yan. Online robust pca via stochastic optimization. In *Advances in Neural Information Processing Systems*, 2013.

[37] Cesar Ferri, Peter Flach, and Jose Hernandez-Orallo. Learning decision trees using the area under the roc curve. In *International Conference on Machine Learning (ICML)*, 2012.

[38] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 2003.

[39] Wei Gao, Rong Jin, Shenghuo Zhu, and Zhi-Hua Zhou. One-pass AUC optimization. In *International Conference on Machine Learning (ICML)*, 2013.

[40] Wei Gao and Zhi-Hua Zhou. On the consistency of AUC pairwise optimization. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[41] James E. Gentle. *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer, 2007.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[43] Neil Gordon, David Salmond, and Adrian Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings*, 1993.

[44] William Greene. *Econometric Analysis*. Prentice Hall, 2011.

[45] San Gultekin and John Paisley. A collaborative kalman filter for time-evolving dyadic processes. In *2014 IEEE International Conference on Data Mining*, pages 140–149, Dec 2014.

[46] Dong Guo and Xiaodong Wang. Quasi-monte carlo filtering in nonlinear dynamic systems. *IEEE Transactions on Signal Processing*, 2006.

[47] Han Guo, Chenlu Qiu, and Namrata Vaswani. An online algorithm for separating sparse and low-dimensional signal sequences from their sum. *IEEE Transactions on Signal Processing*, 2014.

[48] James D. Hamilton. *Time Series Analysis*. Princeton University Press, 1994.

[49] Fang Han and Han Liu. Transition matrix estimation in high dimensional time series. In *International Conference on Machine Learning*, 2013.

[50] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[51] Simon Haykin. *Communication Systems*. John Wiley & Sons, 2008.

[52] Jun He, Laura Balzano, and Arthur Szlam. Incremental gradient on the grassmannian for online foreground and background separation in subsampled video. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[53] Jose M. Hernandez-Lobato, Yingzhen Li, Mark Rowland, Daniel Hernandez-Lobato, Thang D. Bui, and Richard E. Turner. Black box alpha divergence minimization. In *International Conference on Machine Learning*, 2016.

[54] Tom Heskes and Onno Zoeter. Expectation propagation for approximate inference in dynamic bayesian networks. In *Uncertainty in Artificial Infelligence*, 2002.

[55] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[56] Xia Hong, Sheng Chen, and Chris J. Harris. A kernel-based two-class classifier for imbalanced data sets. *IEEE Transactions on Neural Networks*, 2007.

[57] Roger A. Horn and Charles B. Johnson. *Matrix Analysis*. Cambridge University Press, 2012.

[58] Junjie Hu, Haikin Yang, Michael Lyu, Irwin King, and Anthony So. Kernelized online imbalanced learning with fixed budgets. In *Association for Advancement of Artificial Intelligence (AAAI)*, 2015.

[59] Junjie Hu, Haikin Yang, Michael Lyu, Irwin King, and Anthony So. Online nonlinear auc maximization for imbalanced data sets. *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[60] John C. Hull. *Options, futures, and other derivatives*. Pearson, Prentice Hall, 2006.

[61] Kazufumi Ito and Kaiqi Xiong. Gaussian filters for nonlinear filtering problems. *IEEE Transactions on Automatic Control*, 2000.

[62] Satish G. Iyengar, Pramod K. Varshney, and Thyagaraju Damarla. A parametric copula-based framework for hypothesis testing using heterogeneous data. *IEEE Transactions on Signal Processing*, 2011.

[63] Bin Jia, Ming Xin, and Yang Cheng. High-degree cubature kalman filter. *Automatica*, 2013.

[64] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 1999.

[65] Simon K. Julier and Jeffrey K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 2004.

[66] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[67] Majdi Khalid, Indrakshi Ray, and Hamidreza Chitsaz. Confidence-weighted bipartite ranking. In *Advanced Data Mining and Applications*, 2016.

[68] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2014.

[69] Wouter M. Koolen, Alan Malek, Peter L. Bartlett, and Abbasi-Yadkori Yasin. Minimax time series prediction. In *Advances in Neural Information Processing Systems*, 2015.

[70] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug 2009.

[71] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, 2001.

[72] Augustin Lefevre, Francis Bach, and Cedric Fevotte. Online algorithms for nonnegative matrix factorization with the itakura-saito divergence. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2011.

[73] Xiao Rong Li and Vesselin P. Jilkov. Survey of maneuvering target tracking: III. measurement models. In *International Symposium on Optical Science and Technology*, 2001.

[74] Xiao Rong Li and Vesselin P. Jilkov. Survey of maneuvering target tracking. part i. dynamic models. *IEEE Transactions on Aerospace and Electronic Systems*, 2003.

[75] Xiao Rong Li and Vesselin P. Jilkov. Survey of maneuvering target tracking. part v. multiple model methods. *IEEE Transactions on Aerospace and Electronic Systems*, 2005.

[76] Minlong Lin, Ke Tang, and Xin Yao. Dynamic sampling approach to training neural networks for multiclass imbalance classification. In *IEEE Transactions on Neural Networks and Learning Systems*, 2013.

[77] Daniel J. Lingenfelter, Jeffrey A. Fessler, Clayton D. Scott, and Zhong He. Asymptotic source detection performance of gamma-ray imaging systems under model mismatch. *IEEE Transactions on Signal Processing*, 2011.

[78] Chenghao Liu, Steven C. H. Hoi, Peilin Zhao, and Jianling Sun. Online arima algorithms for time series prediction. In *Association for the Advancement of Artificial Intelligence*, 2016.

[79] Lester Mackey, David Weiss, and Michael I. Jordan. Mixed membership matrix factorization. In *International Conference on Machine learning*, 2010.

[80] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 2010.

[81] Peter S. Maybeck. *Stochastic models, estimation, and control*. Academic Press Inc., 1982.

[82] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: a view from the trenches. In *Conference on Knowledge Discovery and Data Mining (KDD)*, 2013.

[83] Jiali Mei, Yohann De Castro, Yannig Goude, and Georges Hebrail. Nonnegative matrix factorization for time series recovery from a few temporal aggregates. In *International Conference on Machine Learning*, 2017.

[84] Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *Uncertainty in Artificial Intelligence*, 2001.

[85] Thomas P. Minka. Power ep. Technical report, 2004.

[86] Naseer Mohammadiha, Paris Smaragdis, Ghazaleh Panahandeh, and Simon Doclo. A state-space approach to dynamic nonnegative matrix factorization. *IEEE Transactions on Signal Processing*, 2015.

[87] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT Press, 2012.

[88] Mahesan Niranjan. Sequential tracking in pricing financial options using model based and neural network approaches. In *Neural Information Processing Systems*, 1997.

[89] John Paisley, David M. Blei, and Michael I. Jordan. Variational bayesian inference with stochastic search. In *International Conference on Machine Learning*, 2012.

[90] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing*, 2014.

[91] Mert Pilanci and Martin J. Wainwright. Iterative hessian sketch: Fast and accurate solution approximation for constrained least-squares. *Journal of Machine Learning Research*, 2016.

[92] H. Vincent Poor. *An Introduction to Signal Detection and Estimation*. Springer, 1998.

[93] Ali Rahimi and Ben Recht. Random features for large scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[94] Ali Rahimi and Ben Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[95] Benjamin Recht, Maryam Fazel, and Pablo A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 2010.

[96] Christian Robert and George Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[97] Sam Roweis and Zoubin Ghahramani. A unifying review of linear gaussian models. *Neural Computation*, 1999.

[98] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, 2007.

[99] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using Markov Chain Monte Carlo. In *International Conference on Machine Learning*, 2008.

[100] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *International Conference on World Wide Web*, 2001.

[101] Robert H. Shumway and David S. Stoffer. An approach to time series smoothing and forecasting using the EM algorithm. *Journal of Time Series Analysis*, 1982.

[102] Bruno Sinopoli, Luca Schenato, Massimo Franceschetti, Kameshwar Poolla, Michael I. Jordan, and Shankar S. Sastry. Kalman filtering with intermittent observations. *IEEE transactions on Automatic Control*, 2004.

[103] John Sun, Dhruv Parthasarathy, and Kush Varshney. Collaborative kalman filtering for dynamic matrix factorization. *IEEE Transactions on Signal Processing*, 62(14):3499–3509, 2014.

[104] John Sun, Kush Varshney, and Karthik Subbian. Dynamic matrix factorization: A state space approach. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012.

[105] Jiexiong Tang, Chenwei Deng, and Guang-Bin Huang. Extreme learning machine for multilayer perceptron. *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[106] Jing Teng, Hichem Snoussi, Cedric Richard, and Rong Zhou. Distributed variational filtering for simultaneous sensor localization and target tracking in wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 2012.

[107] Joel Tropp. An introduction to matrix concentration inequalities. *Foundations and Trends in Machine Learning*, 2015.

[108] Malik Tubaishat and Sanjay Madria. Sensor networks: An overview. *IEEE Potentials*, 2003.

[109] Rudolph van der Merwe, Arnaud Doucet, Nando de Freitas, and Eric Wan. The unscented particle filter. In *Neural Information Processing Systems*, 2000.

[110] Aad W. Van der Vaart. *Asymptotic Statistics*. Cambridge University Press, 1998.

[111] Jaco Vermaak, Neil D. Lawrence, and Patrick Perez. Variational inference for visual tracking. In *Computer Vision and Pattern Recognition*, 2003.

[112] Martin Wainwright and Michael Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2), 2008.

[113] Greg Welch and Gary Bishop. An introduction to the Kalman filter. Technical report, Chapel Hill, NC, US, 1995.

[114] Liangbei Xu and Mark A. Davenport. Dynamic matrix recovery from incomplete observations under an exact low-rank constraint. In *Advances in Neural Information Processing Systems*, 2016.

[115] Sun Yi, Daan Wiestra, Tom Schaul, and Jurgen Schmidhuber. Stochastic search using the natural gradient. In *International Conference on Machine Learning*, 2009.

[116] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S. Dhillon. Temporal regularized matrix factorization for high-dimensional time series prediction. In *Advances in Neural Information Processing Systems*, 2016.

[117] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *International Conference on Machine Learning (ICML)*, 2004.

[118] Peilin Zhao, Steven C. H. Hoi, Rong Jin, and Tianbao Yang. Online AUC maximization. In *International Conference on Machine Learning (ICML)*, 2011.

[119] Renbo Zhao, Vincent Y. F. Tan, and Huan Xu. Online nonnegative matrix factorization with general divergences. In *International Conference on Artificial Intelligence and Statistics*, 2016.

[120] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology*, 2005.