Bayesian Approaches to Rare Event Prediction in Multivariate Time Series

Ryan Turner
Department of Engineering
University of Cambridge

First Year Report
Second Edition
November 19 2008
Acknowledgements

I would like to thank Zoubin Ghahramani and Carl Rasmussen for advice on the content of the report; Steve Bottone for information on the DataPath problem and the data sets; and Shakir Mohamed, Finale Doshi, and Marc Deisenroth for help with \LaTeX{} and pdf conversion.
Abstract

The problem of rare events in large multivariate time series is explored within the context of a condition monitoring and failure prediction problem. A data set from DataPath’s MaxView software is used. We establish a baseline for prediction on this data set. This report examines feature extraction and similarity metrics for multivariate time series. The aspect of rare events is most relevant in sections discussing methods for algorithm performance evaluation and similarity metrics, which are used for exploratory data analysis. The modeling sections pay attention to the sensitivity of the methods to outliers and non-Gaussian behavior. After reviewing standard time series models in the linear Gaussian framework, extensions are explored; common extensions include the ability to handle rare or non-Gaussian events, non-parametric versions, and factorial models. I propose various extensions to standard time series models, which are tailored to the failure prediction problem.
Contents

1 Introduction

2 Overview
   2.1 Overview of DataPath Project
   2.2 Overview of Rarity
   2.3 Overview of Temporal Data

3 DataPath Data Set
   3.1 Data Representations
      3.1.1 SQL Representation
      3.1.2 Mathematical Waveform Representation
      3.1.3 The Feature Matrix
         3.1.3.1 Alarm Level Representation
         3.1.3.2 Alarm Edge Representation
         3.1.3.3 Time to Fault Representation
   3.2 Exploratory Data Analysis
      3.2.1 Raw Data
      3.2.2 Similarity Metrics
         3.2.2.1 Lag 0 Mutual Information
         3.2.2.2 Lagged Mutual Information
   3.3 Other Challenges
   3.4 Baseline Results
   3.5 Conclusion

4 Feature Extraction for Time Series
   4.1 Temporal Data Analysis
      4.1.1 Similarity Metrics
   4.2 Principal Component Analysis
4.2.1 Eigenface Experiment ........................................... 46
4.2.2 Random Square Experiment ................................... 49
4.2.3 PCA on the DataPath Data Set ................................. 52
4.3 Clustering .......................................................... 55
4.4 Probabilistic Matrix Factorization ................................. 57

5 **Standard Time Series Models** .................................. 59
  5.1 Autoregressive Models ........................................... 59
  5.2 Linear Dynamical Systems ....................................... 64
    5.2.1 Model Setup .................................................. 65
    5.2.2 An Illustrative Example ...................................... 65
    5.2.3 Inference ..................................................... 67
    5.2.4 Learning ..................................................... 68
    5.2.5 Extended Kalman Filter ...................................... 69
    5.2.6 Particle Filtering ............................................. 72
  5.3 Hidden Markov Models ........................................... 73
    5.3.1 Model Setup .................................................. 73
    5.3.2 Baum-Welch Algorithm ...................................... 75
      5.3.2.1 Forward-Backward Algorithm ............................ 75
      5.3.2.2 Parameter Updates ....................................... 76
      5.3.2.3 Numerical Difficulties ................................... 77
  5.4 Rare Events ..................................................... 78

6 **Extensions to Classic Time Series Models** ....................... 81
  6.1 Switching Kalman Filter ........................................ 81
  6.2 Novelty Detection with the SKF ................................. 84
  6.3 Factorial Hidden Markov Model .................................. 86
  6.4 Infinite Hidden Markov Model .................................... 87
  6.5 Phase Type Distributions ....................................... 88

7 **Research Ideas** .................................................. 89
  7.1 Latent Cause Model ............................................. 89
    7.1.1 Autoregressive Causes ...................................... 89
    7.1.2 Hidden Causes ............................................... 91
    7.1.3 Non-Parametric Version: Infinite Causes ................... 91
  7.2 Binary Probabilistic Matrix Factorization ....................... 92
# List of Figures

1.1 Comparison of donuts and donut holes. ........................................... 2

2.1 Screen shot of the MaxView software from Bottone et al. (2008). It graphically represents many components in the system and simply represents the state of the system. However, it does not do any statistical analysis or prediction. ................................................................. 6

2.2 Summary of ROC and PR metric from Davis and Goadrich (2006). .... 9

2.3 Summary of Allen’s relations from Morchen (2006). ......................... 12

3.1 Abstraction for the fault alarms ....................................................... 15

3.2 Abstraction for the analog sensors. Note that we observe a range of values over time. ................................................................. 16

3.3 Visualization for all the fault log data. White represents a fault, red an alarm, and black is the normal state. .......................................... 20

3.4 Empirical CDF for number of alarm occurances. This CDF plots the distribution over the number of times a variable transitions from normal to alarm or fault. Each variable is treated as a sample under this distribution. 21

3.5 Empirical CDF of inter-arrival times for alarms. The inter-arrival times are defined as the time between the transitions of a particular variable from normal to alarm or fault. The inter-arrival times for all the variables are combined in this CDF. Each transition out of the normal state in the data set is treated as a sample in this distribution. ......................... 22

3.6 Empirical CDF over mean inter-arrival times for alarms. Alarm inter-arrival times are defined the same here as in Figure 3.5, but here there is one sample for each variable. Each sample is the mean inter-arrival time on a given variable. ........................................... 23
3.7 Fit of inter-arrival time data to exponential distribution. It has a log-likelihood of $-9.4561 \times 10^5$. It has a fitted mean of 49073 seconds or about 13.5 hours. If the data fit an exponential perfectly then it would follow the dashed line. Clearly, the exponential distribution under-estimates the long tail of the data.

3.8 Fit of inter-arrival time data to Weibull distribution. It has a log-likelihood of $-6.6225 \times 10^5$. The parameters are $a = 1063.2$ (scale) and $b = .3475$ (shape). If the data fit a Weibull perfectly then it would follow the dashed line. Although it fits the data better than the exponential it still under-estimates the activity at the tails. Perhaps a mixture distribution would be best. Given that this is an aggregate of different alarms a mixture model may be appropriate.

3.9 Empirical CDF of alarm durations. The duration of an alarm is the defined as the time between when there is a transition from the normal state to a non-normal state and when the alarm transitions back to the normal state.

3.10 Heatmap for mutual information matrix. Information here is measured in nats.

3.11 Close up of sorted mutual information matrix in high mutual information area. Information here is measured in nats.

3.12 A graph between nodes that have high mutual information ($> .35$ nats). The term “Latched” has been abbreviated with L. The system contains disjoint and almost fully connected cliques.

3.13 Graphical model for standard Gaussian mixture model (single task)

3.14 Graphical model multi-task Gaussian mixture model over $K$ tasks

3.15 Performance for linear regression on Dew Point Fault - ETI Dehydrator ADH-2A COM (variable 5). It is interesting that the PR curve goes up and then down instead of a steady downward descent. Usually, as you set the threshold lower the precision becomes lower because the extreme positive predictions become polluted with the moderate ones. Because having a moderate threshold is better it suggests the extreme positive predictions are actually poor indicators of an alarm happening. The moderate ones appear to be a better indicator for this variable. Most likely something changed in the system state between training and test.
3.16 Performance for linear regression on Dew Point Fault - ETI Dehydrator ADH-2A COM (variable 5) with a dimensionality reduced version of the data .......................................................... 38
3.17 Performance for logistic regression on Dew Point Fault - ETI Dehydrator ADH-2A COM (variable 5) with a dimensionality reduced version of the data .......................................................... 39

4.1 Graphical model for pPCA. $y$ refers to the observed data, $x$ is the low dimensional projection of $y$. $C$ is the matrix describing the linear manifold between $x$ and $y$. $R$ is the covariance matrix on the noise for $y$. .......................... 45
4.2 Original faces used for the eigenface experiment from Serrano (2004) . . . 47
4.3 Principal components of the original faces, Figure 4.2, in the eigenface experiment. All of these images are orthogonal to each other when reshaped as a vector. ......................................................... 48
4.4 A typical sample in the random square experiment ................................. 49
4.5 RMSE values for inferred position by real position. For instance, the color at location (5,5) is the RMSE error between (5,5) and the inferred location by the algorithm when the square is actually located at (5,5). The error is quite low in most cases. Off by just one or two pixels, which could be explained by round off error given that the true location is always an integer. However, the performance is quite bad in the corners. This is to be expected given that for all corner but the upper left most of the square will be off the image. Even in the upper left corner the performance is a bit lower. ................................................................. 50
4.6 Reconstruction for random square back of Figure 4.4 into the original 400D space ............................................................ 51
4.7 Features for DataPath time series. All 388 alarms have been reduced down to 25 principal components. ................................. 52
4.8 reconstruction for DataPath time series ........................................... 53
4.9 Projection of the time points to two dimensions. Places where alarm 9 is on are shown in red. Note that the alarm being on in a time interval is not the same as it going from the off state to on in that time interval. . 54
4.10 Graphical model for PMF from Salakhutdinov and Mnih (2008) .......... 57

5.1 Graphical model for AR(2) model. Each observation is a random variable that is dependent on the last two values in the time series. .............. 59
5.2 Graphical model for a MA(2) model. The $y$ variables are black nodes because they are deterministic given the random movements, epsilon.

5.3 Graphical model for an ARMA(1, 2) model

5.4 Graphical model for ARCH model

5.5 Graphical model for GARCH model

5.6 Graphical model for the LDS from Ghahramani and Hinton (1998)

5.7 Simulation of Kalman filter on an LDS

5.8 Close up of simulation for orbit example

5.9 Zoom out of simulation for orbit example

5.10 Second run of orbit experiment

6.1 Graphical model for a simple SKF from Ghahramani and Hinton (1998). The continuous valued hidden state is represented by $x_t$. The observations are $y_t$ and $s_t$ represents the output mode of the system.

6.2 Graphical model for more complex SKF from Ghahramani and Hinton (1998)

6.3 Graphical model for SKF with external inputs from Ghahramani and Hinton (1998)

6.4 Graphical model for SKF with autoregressive outputs from Ghahramani and Hinton (1998)

6.5 Graphical model for Ghahramani and Hinton’s SKF from Ghahramani and Hinton (1998)

6.6 Alternative model for Ghahramani and Hinton’s SKF from Ghahramani and Hinton (1998). One can imagine that each $X$ box is evolving independently according to a linear system. Likewise, $S$ is evolving independently according to a Markov chain. The output $Y$ is the result of using $S$ as a selector on the different $X$ chains and then adding some noise.

6.7 Distribution over the observation. Each Gaussian represents the distribution on the output conditioned on the system being in a certain state, $s$. The dashed line represents the output given it is in the X-factor state.

6.8 Graphical model for an FSKF used on condition monitoring in a medical application (Williams et al., 2006). It illustrates how an X-factor can be used in practice.

6.9 Graphical model for an FHMM from Ghahramani and Jordan (1997)

7.1 Representation of noisy OR process for alarms

7.2 Graphical model for BPMF
8.1 Gantt chart for PhD research. Level 1 of the Latent Cause Model refers to the autoregressive version while level 2 refers to hidden causes. 

A.1 Box plot for duration of alarms over all alarms. Time is measured in log10 scale.

A.2 Box plot for inter-arrival times of alarms over all alarms. Note that time is measured in log10 seconds.
Chapter 1

Introduction

Machine Learning is the study of algorithms whose performance improves with increased exposure to data. Most computer programs do not change no matter how much data they process; their entire functionality is specified in advance by the programmer. A machine learning algorithm, by contrast, will become more intelligent as more data is processed. Most machine learning algorithms do not exist in a vacuum. They are usually tied to a given application area, which utilizes specialized domain knowledge. Some of these areas include speech applications, computer vision, bioinformatics, finance, and the focus of my research, condition monitoring. To make things more concrete, let us suppose an example. Culinary metaphors\(^1\) are common in machine learning so a bakery example will be used. This example will use computer vision as the domain area.

Suppose there is an assembly line in a bakery that is pumping out donuts and donut holes; they are mixed together on the same conveyor belt. The owner would like to sort the donuts into one bin and the donut holes into another. To do this a camera is placed above the conveyor belt. The data is fed into a computer. A programmer is assigned the task of recording the data. The programmer knows how to take the results and store them in a database, possibly make a web interface to the database, interface to the camera, and so on.

There is one pesky function this programmer does not know how to write. How do we take this array of 1000 by 1000 pixels, the image, and output the number of donuts and the number of donut holes? Computer vision with machine learning in the background is called upon to come to the rescue. Computer vision techniques will help extract important statistics on the image and do various transforms, such as edge detection, that make the

\(^1\)The Chinese Restaurant Process (CRP) and Indian Buffet Process (IBP) are major developments in machine learning that use culinary metaphors.
image easier to work with. The resulting data will be very noisy. Edge detectors do not perform perfect, or even decent, object segmentation. Machine learning will have to come in on the back-end.

The task of identifying an object as either a donut or a donut hole is called classification. In this case, it is binary classification. One could also include a third type, cookies, and it would still be classification. If one had to identify certain continuous valued statistics on the objects, such as the diameter of the donuts holes or their estimated weight, then it would be a regression problem.

Machine learning algorithms are typically trained via one data set, the training set, and tested by a test set. Testing the performance on the training set would be misleading as good performance on a training set does not imply generalization. After all, one could just memorize the correct output for a training set. Presumably, in this case, supervised learning would be used. A set of input images is provided along with the correct output for each image. Presumably, the training set here would have been made by a human manually labeling the images.

In many cases, the training points are assumed to be independent. For example, the class of one object on the conveyor belt does not provide information about the next. However, if the donut and donut hole machines output in bursts then this independence property clearly no longer holds. A time series model has to be applied. If the classification of one object is independent of all the ones before given the immediately previous one then it satisfies the Markovian property.

It is also possible for data imbalance or relative rarity to occur. If 90% of the objects were donuts and only 10% were donut holes, the data set would be imbalanced. Many machine learning algorithms are not designed to work well with this. There could also be few examples altogether of donuts holes, maybe only 10, which would constitute absolute
rarity. When this is the case the algorithm will have to be based on a model, which lets it parameters gain statistical strength wherever they can. In other words, one cannot use a naive model that throws away information for simplicity. For instance, it will be more important to look at the time series information and see what conditions led to the appearance of donut hole. If the few donut holes that rolled by occurred in a burst, it will be very informative to utilize that information.

The focus of this report will relate to a condition monitoring application involving satellite earth terminals. To do this, time series models that model imbalanced and rare data will be explored. The models will be created primarily using concepts from Bayesian statistics.

The first part of the report focuses on the data set at hand and the challenges it might present. In Chapter 2 the report will give an overview of the three main aspects of the DataPath problem. Firstly, there is an overview of the data set itself and and what kind of phenomenon might be observed. Then there is an overview of rarity and temporal data, the two other aspects of the problem. In Chapter 3, the report delves into more detail on the data set. Different ways of casting the data in mathematical form are explored. Different types of feature matrices are presented as well. Then there is an exploratory data analysis. In addition to plotting the raw data and some descriptive statistics various similarity metrics are explored. Their purpose is to detect if we can pick up any signal in the data and what form the relationships take. The performance of simple algorithms is presented in order to provide a baseline for future algorithms.

The following chapters focus more on machine learning methods than on the data set itself. In Chapter 4, there is an overview of some feature extraction methods. The report also presents examples for how they can be used. In Chapter 5, there is a review of the standard models for time series: ARMA, LDS, and HMMs. In Chapter 6, the models are extended to make them more flexible, but also less tractable. Finally, in Chapter 7, research ideas are presented. The one of primary focus is the latent cause model, where system failures are viewed to have been the result of a number of other causes using a noisy OR framework. Other ideas involve extensions for matrix factorization and dimensionality reduction.
Chapter 2

Overview

2.1 Overview of DataPath Project

DataPath specializes in monitoring and control software and satellite earth terminal units. Its software, MaxView, is used for mission-critical applications in network management. It is usually used to monitor satellite earth terminals and log the results of various measurements. MaxView presents users with a virtual control board for monitoring on a computer. It is designed so operators with minimal programming experience can customize the GUI, automate certain network tasks, and write drivers that interface the software to new network hardware. The basic premise of the DataPath problem is to add a feature to the MaxView software that can give operators a warning when a fault is likely to occur. Adding some prediction capability could potentially serve as value added to the software. This type of problem falls in an area known as condition monitoring. It also involves novelty detection because many of the failures may be rare.

A technician will monitor the state of the system by watching the measurements on the screen, as seen in Figure 2.1. However, the display can only show so many variables at once. Likewise, a technician can only keep track of a limited number of variables at once and comprehend a limited number of relationships. If an algorithm can help predict when something is out of the ordinary then it adds to the effectiveness of the software and the productivity of any technician working on it.

In addition, the quantity of the data collected by these systems is much larger than what can be acquired through laboratory QA testing. Therefore, any patterns could be of use to quality assurance (QA) as well. For instance, if the data can help estimate a conditional probability distribution for failure of an amplifier conditioned on temperature and voltage the QA people could be very interested.
Figure 2.1: Screen shot of the MaxView software from Bottone et al. (2008). It graphically represents many components in the system and simply represents the state of the system. However, it does not do any statistical analysis or prediction.
For example, a typical sub-system is a traveling wave tube high-power amplifier (TWTA). It may contain measurements at the following points:

- Helix Voltage (kV)
- Helix Current (mA)
- Cathode Filament Current (A)
- Tube Temp (C)
- Power Supply Temp (C)

The cathode current will generally slowly degrade before a TWTA fails. The helix current will also ramp up immediately before failure (Bottone et al., 2008). The goal is to create a system that can automatically determine these types of relationships. One of the potential challenges is that phenomena could occur on many different time scales. For instance, one fault predictor could only be informative one minute before failure. Another might have a subtle trend over a month which is predictive.

The data is stored in an SQL database. They are generally on the order of 100 MB of data per month. However, this varies from one installation to the next.

2.2 Overview of Rarity

A key aspect of the DataPath problem is rarity and class imbalance. Alarms occur very few times in general. There are also many more examples of times when an alarm does not appear than when it does.

Rarity presents multiple problems in machine learning. The focus here will be on rarity in classification. There are two types of rarity: absolute rarity and relative rarity (Weiss., 2004). Absolute rarity is the case when there are very few examples of when a class is even observed. This is obviously difficult because there are not very many cases to train to. The other kind of rarity is relative rarity, also known as class imbalance; it is when a certain class for a classifier occurs quite seldom, perhaps < 1% of the time, but it may have many cases because there is an abundance of total cases. Relative rarity can still cause problems for evaluation and for naive algorithms. For instance, in classification, the absolute classification error rate is a poor loss function in evaluation when data is rare. If a class B appears 1% of the time then always picking class A will
yield an accuracy of 99%. Other metrics, such as the balanced error rate can be used. The balanced error rate (BER) in this example will be 50%.

\[
\text{Error Rate} = P(\text{misclassification}) = P(\text{label positive}|\text{really negative})P(\text{really negative}) + P(\text{label negative}|\text{really positive})P(\text{really positive})
\]

\[
\text{BER} = \frac{1}{2}(P(\text{label positive}|\text{really negative}) + P(\text{label negative}|\text{really positive}))
\]

If one has a particular loss function that has a justification for the problem it can be used as a metric as well.

\[
E[\text{Loss}] = L_1 P(\text{label positive}|\text{really negative})P(\text{really negative}) + L_2 P(\text{label negative}|\text{really positive})P(\text{really positive})
\]

$L_1$ is the cost of classifying a negative example as positive and $L_2$ is the cost of classifying a positive example as negative. Here it is assumed there is no cost to correct classification.

Another performance metric is the receiver operating characteristic (ROC). This curve can be used if the classifier outputs a probability measure as its output. The ROC curve plots the true positive rate versus the false positive rate (sensitivity versus 1 – specificity in classical statistics terminology) over different classification thresholds. To summarize the ROC curve, the area under the curve (AUC) is often used. The AUC can be interpreted as the probability a randomly chosen positive example is ranked higher than a randomly chosen negative example (Fawcett, 2006). Therefore, the AUC is similar to the balanced error rate in that it does not penalize minority classes. The AUC is different because it is used on algorithms that have probabilistic output.

Another metric besides the ROC curve is the precision-recall (PR) curve. Recall is defined the same as the true positive rate. Precision is defined as the probability an example actually is positive given it was labeled positive. It inverts the condition probability in true positive rate. A summary of the metrics is in Figure 2.2.

\[
\text{recall} = P(\text{label positive}|\text{really positive}) \quad (2.1)
\]

\[
\text{precision} = P(\text{really positive}|\text{label positive}) \quad (2.2)
\]

\[
\text{true positive rate} = P(\text{label positive}|\text{really positive}) \quad (2.3)
\]

\[
\text{false positive rate} = P(\text{label positive}|\text{really negative}) \quad (2.4)
\]
Figure 2.2: Summary of ROC and PR metric from Davis and Goadrich (2006).

Example ROC and PR curves can be seen in Figure 3.15. A perfect classifier will have an ROC curve in the upper left hand corner. By contrast, a perfect PR curve will be in the upper right hand corner. With imbalanced data sets it is especially important to note the difference between precision and the false positive rate. The false positive rate looks at the number of positive labels in the true negative class. If the negative class is very large then even if most of the positive labels are indeed negative examples the false positive rate will be low. In this case, by definition, the precision will be low. (Davis and Goadrich, 2006)

Relative rarity can turn into absolute rarity when considering combinations, which effectively puts the observations in a higher dimensional space. Recommender systems for online shopping deal with this problem when posed with recommending combinations of items. Items may not be absolutely rare marginally, but when considering their joint behavior they are. A similar situation can occur when estimating N-grams in natural language applications where many words are not rare on their own, but when combinations in a sentence begin to be considered they become much rarer. By the time one moves to 3-grams or 4-grams the empirical frequencies become very poor estimates of the probabilities. For instance, the probability of “Thursday that they had” will be much harder to estimate via relative frequency than the marginal probability of the words individually. More structured learning is required in these cases (Teh, 2006).

Absolute rare cases are obviously more susceptible to noise given there are less examples to learn from. Overfitting becomes much more of a concern with rare data; high levels of noise in the data exacerbates the problem. When the ratio of data to parameters or noise is low overfitting can easily occur.

For algorithms that have trouble with data imbalance, such as support vector machines (SVMs) (Akbani et al., 2004), sampling methods are sometimes used as heuristics. One
method is to under sample the data; this method throws away many of the points in
the majority class to attain class balance. This is undesirable as one is not utilizing a
large portion of their data. Over sampling involves duplicating points in the minority
class to attain class balance. This type of replication can lead to overfitting. There are
other more sophisticated sampling strategies. For example, the synthetic minority over-
sampling technique (SMOTE) places the duplicated data points in-between two members
of the minority class. Ensemble methods will create multiple data subsets to train on. For
instance, Chan and Stolfo (1998) splits an imbalanced data set into multiple balanced
data sets. Each balanced data set includes all the minority points and some of the
majority data points. Each data point in the majority class is used in at least one subset.
An SVM is trained on each data subset. The recommendation of each SVM is used to
train a mixture of experts model. All these methods are unnecessary for a generative
model that deals with the imbalance explicitly by estimating the class probability rather
than giving a hard classification. It will extract all the information present in the original
data points so there is no need for sampling.

2.3 Overview of Temporal Data

Another key aspect of the DataPath problem is the temporal nature of the problem.
This distinguishes it from an ordinary imbalanced classification problem. Therefore, it is
important to cover the key concepts in temporal data.

Temporal data comes in a variety of forms. It is important to distinguish between
a time series and a sequence. A time series is a set of time points, which usually have
an associated value. Time points are single points along the real line, which is usually
discretized to some level such as seconds or milliseconds. A sequence is a set of ordered
observations. A classic example of a sequence is genome data, where the data is ordered
but doesn’t have particular time stamps. Stock market data forms a time series since
each stock quote or trade has a real time stamp associated with it. Sampling on time
series is usually uniform, but this is not always the case.

A pair of time points can form an interval. One can construct an interval series via a
set of non-overlapping time intervals \(^1\) An interval sequence has no such restrictions. A

\(^1\)In many real databases this is often violated as no one is checking for inconsistencies in the data. It
is surprising how often the SQL command \( \text{SELECT * FROM foo WHERE endTime < startTime} \) returns
results.
contiguous interval series is an interval series where there are no gaps between the time intervals. (Morchen, 2006)

A numeric time/interval series/sequence has numeric values at the time points or during the intervals. A symbolic time/interval series/sequence consists of nominal data at the time points or intervals. The genome data is a nominal sequence because the data points are either adenine (A), cytosine (C), guanine (G) or thymine (T). The stock data is a numeric time series as the values are real numbers (at least to the nearest cent). The fault log in a MaxView database is a symbolic interval series before any feature extraction takes place. The values are either normal, alarm, or fault. They occur on intervals until the fault is cleared and it is a series since the interval’s starts and ends are recorded to the second. The analog log is a numeric time series because the readings are only at single points in time. Obviously, temporal data can be multivariate or univariate. The fault log and the analog log are multivariate because there are various data streams occurring simultaneously.

Data mining research is sometimes focused on finding rules for prediction in temporal data. To construct these rules they invent various operators. For time points relations consist of before, equals, and after. For rule based systems, two time points are rarely tested for exact equality. Instead, they check if the time points are within a small time delta of each other. For intervals there is a set of relations known as Allen’s relations or operators: before, meets, overlaps, starts, during, finishes, and equals. They have the inverse operators: after, met by, overlapped by, started by, contains, and finished by, respectively. Equals is an inverse with itself. The relations are represented visually in Figure 2.3.

Again when building rule based systems a small window of granularity is added around equality constraints on the end points to add robustness. Even if one is not implementing a rule based system these distinctions are important to have in mind when designing models and feature extractors. Furthermore, they are useful for software engineering as they provide good test points to consider temporal routines on.

It is important to note the difference between filtering, smoothing, and prediction. Filtering is doing inference on the hidden state given past information

\[ p(x_t|y_{1:t}) \]

Here \( x_t \) refers to the hidden state of a system at time \( t \); \( y_{1:t} \) refers to the observations of the system from time 1 to \( t \). In smoothing we want to infer the hidden state given all the information we have
In prediction we care about the future observed state

\[ p(x_t|y_{1:T}), \ T > t \]

In real time applications, such as navigating the position of a mobile robot, filtering is the approach which is used because it must be causal. If that robot were taking pictures along its route and we wanted to geolocate those photographs then future information could be incorporated. The calculation would be

\[ E[\text{true location at time } t| \text{noisy readings on location from 1 to } T] \]

If one was using a Kalman filter for predicting financial time series then the prediction task would obviously be the approach to take

\[ p(\text{return at time } t + 1| \text{returns at times 1 to } t) \]

The observed state not the underlying dynamics is of interest in this case. When doing inference for the purpose of learning, smoothing will be used since we want to incorporate all available information.
Chapter 3

DataPath Data Set

3.1 Data Representations

The DataPath data is stored in an SQL database. The database is composed of several tables, but we are primarily concerned with two data sources: the fault log and the analog log. Before any machine learning techniques can be applied the data in the database must be put in mathematical form. Each possible fault and each analog sensor will be represented as a function over time. Each alarm in the fault log represents an interval series if the end times are considered; it will be normal point process if alarm end times are thrown away. The analogs are a time series with non-uniform sampling. Furthermore, time will be discretized in order to create feature matrices. There are multiple approaches for discretization. After creating feature matrices we explore various statistics of the data to look for useful signals in the data that may point in useful directions.

\[
\begin{array}{|l|}
\hline
\text{SQL Tables} \\
\text{Mathematical Waveform} \\
\text{Feature Matrix} \\
\hline
\end{array}
\]

Table 3.1: Diagram showing the three levels of abstraction.

3.1.1 SQL Representation

There are several tables in a SQL database from MaxView. The ones we are concerned with are the fault log and analog log.

The fault log is the table of primary interest. The log variable is when a given alarm starts. computer_id and broker_id are related to how the data is reported and
Table 3.2: Five random entries from the fault_log table. The columns marked IGNORED are considered to be irrelevant. Each unique combination in the columns marked VARIABLE represents a variable, or device index, in the mathematical representation.

<table>
<thead>
<tr>
<th>log</th>
<th>broker_id</th>
<th>computer_id</th>
<th>device_id</th>
<th>field_id</th>
<th>val</th>
<th>acknowledged</th>
<th>masked</th>
<th>lev</th>
<th>cleared</th>
<th>log_millis</th>
</tr>
</thead>
<tbody>
<tr>
<td>22/12/2007 14:05:59</td>
<td>1</td>
<td>1094429460</td>
<td>1</td>
<td>1097013846</td>
<td>197</td>
<td>ALARM</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>23/12/2007 15:06:28</td>
</tr>
<tr>
<td>24/12/2007 07:03:48</td>
<td>1</td>
<td>1131323653</td>
<td>273678629</td>
<td>63</td>
<td>7</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: Five random entries from the analog_log. The columns marked IGNORED are considered to be irrelevant. Each unique combination in the columns marked VARIABLE represents a variable, or device index, in the mathematical representation.

<table>
<thead>
<tr>
<th>log</th>
<th>broker_id</th>
<th>computer_id</th>
<th>device_id</th>
<th>field_id</th>
<th>val</th>
<th>log_millis</th>
</tr>
</thead>
<tbody>
<tr>
<td>06/12/2007 20:27:33</td>
<td>1</td>
<td>1131323653</td>
<td>273678629</td>
<td>63</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>19/12/2007 11:18:31</td>
<td>1</td>
<td>1131323653</td>
<td>273678349</td>
<td>63</td>
<td>7.2</td>
<td>0</td>
</tr>
<tr>
<td>18/11/2007 07:39:15</td>
<td>1</td>
<td>1094613087</td>
<td>1096251471</td>
<td>17</td>
<td>73.3</td>
<td>0</td>
</tr>
<tr>
<td>15/12/2007 20:55:38</td>
<td>1</td>
<td>1131323653</td>
<td>273678350</td>
<td>63</td>
<td>7.5</td>
<td>0</td>
</tr>
<tr>
<td>21/12/2007 11:38:05</td>
<td>1</td>
<td>1131323653</td>
<td>273678628</td>
<td>63</td>
<td>3.1</td>
<td>0</td>
</tr>
</tbody>
</table>

The analog_log is also of interest. It logs values that are continuous such as the current in an amplifier or power supply voltage. The fault_log might contain these variables, but only log that a critical threshold has been crossed and needs to be watched, such as temperature. The analog_log will save the state of a variable every time it changes by a certain increment.

The field values for the analog_log are similar to those for the fault_log except that it doesn’t save a cleared time because it saves point samples and not intervals. It is also missing some fields that are not of interest for this problem. Other practical considerations include issues such as cleaning up the val field since it contains units not just the numbers. The database was set-up for reporting to humans and not just for
machines. Note that although we only observe point samples we are really observing a range of possible values for a given range of time. Because if the value left that range it would have triggered an observation.

### 3.1.2 Mathematical Waveform Representation

The next step in processing the data set is developing a mathematical representation for it. The basic abstraction of the data set for the purposes of prediction is in Figures 3.1 and 3.2.

![Example of an alarm interval series in mathematical form](image)

Figure 3.1: Abstraction for the fault alarms

### 3.1.3 The Feature Matrix

The final step in getting the data ready for the algorithms is to represent it in matrix form; almost all machine learning algorithms expect data in matrix form. There are three
Figure 3.2: Abstraction for the analog sensors. Note that we observe a range of values over time.
ways to do this, with the most convenient depending on what type of algorithm you are using.

3.1.3.1 Alarm Level Representation

My first method consists of dividing the time span into 15 minute intervals; however, any time bin could be used. The value for a given alarm in a given interval is the maximum of the level of the alarm during that period. The alarm is considered to last from when it is logged to when it is cleared.

\[ A_{ij} = \max_{T_j \leq t < T_{j+1}} W_i(t) \]

The term st is short for such that. \( A \) represents the alarm matrix. \( W_i \) is the mathematical waveform for the \( i^{th} \) alarm. \( T_j \) is the time at the beginning of the \( j^{th} \) interval. The alarm level representation for the entire data set is visualized as a heatmap in Figure 3.3.

3.1.3.2 Alarm Edge Representation

The second method is the maximum level jumped to during a given interval. So if no alarm went on the value is zero. If an alarm went on it is the level of the alarm that is recorded. If two alarms go on, the one with the highest level is recorded in that interval.

\[ E_{ij} = \begin{cases} 0 & \text{if } \Delta W_i(t) = 0, \forall t, T_j \leq t < T_{j+1} \\ A_{ij} & \text{otherwise} \end{cases} \]

\( E \) is the edge representation matrix. \( \Delta W \) represents the instantaneous change in \( W \).

3.1.3.3 Time to Fault Representation

The third form is to divide the matrix into time points instead of intervals. At each interval we store how long until the next alarm occurs in the value. There are really two matrices here. There is the one we predict from, \( F^{\text{since}} \), which is how long it has been since the last alarm for each variable. The second matrix would store how long until the next fault, \( F^{\text{until}} \), for each alarm and be used as a target matrix in training. This orients the problem more to a regression setting rather than a classification one. It is also similar to what one would use survival analysis for in standard statistics. However, the fact that only one trial occurs for each pair of variables limits the survival analysis options. Survival analysis is harder, although still possible, in situations where the faults
are recurrent (can happen more than once) and we are looking at relationships between variables.

\[
F_{ij}^{\text{until}} = \min t \text{ st } T_j \leq t, \ \Delta W_i(t) > 0 \\
F_{ij}^{\text{since}} = \max t \text{ st } t < T_j, \ \Delta W_i(t) > 0
\]

\(F_{ij}^{\text{until}}\) is the matrix of times until the next fault and \(F_{ij}^{\text{since}}\) is the matrix of times since the last fault.

Structured methods such as an HMM could translate between the two forms of prediction. An HMM would be more suited towards the discretized level state model, but we could translate predictions into expected time to failures by extrapolating out several time steps and taking the expectation of the next time we will see a failure. However, this is not as easy as with a model that looks at fault prediction as a regression problem. Also, to establish a baseline with linear or logistic regression it would be harder, because they do not explicitly model time. So it couldn’t extrapolate as easily.

The calculation of the cumulative distribution function (CDF) for time until a failure, given the probabilities of failures in given intervals, assuming they are independent, is

\[
P(t < 1) = p_1 = 1 - q_1 \\
P(t < 2) = 1 - q_1 q_2 \\
\vdots \\
P(t < \tau) = 1 - \prod_{i=1}^{\tau} q_i
\]

The time to to fault is represented by \(t\). The probability a fault occurs in a window of time \(i\) is \(p_i\) while the probability a fault does not occur in time window \(i\) is \(q_i\). If all the \(p\)'s are equal and the time interval goes to zero, this is the definition of a Poisson point process and the distribution of time to failure is exponential. If the failure probabilities in the intervals are not independent then the conversion is model specific. Once a distribution over time to failure is specified an expectation over the time to fault distribution can be calculated.

It is an open question what most clearly represents the state of the system. A clear representation is a necessity for simple models and helpful for more complex ones. The cleared field is especially suspicious. In some cases the cleared field stays on until a technician turns it off; sometimes it goes off on a timer. In both cases, the cleared field
does not represent anything useful about the state of the system. Ignoring it completely is not a good option because for some alarms it does represent the resolution of a problem.

## 3.2 Exploratory Data Analysis

### 3.2.1 Raw Data

Most of the alarms are off most of the time. Some of them never even occur. There are a few alarms which stay on for very long periods of time. Some of these are latched variables, meaning they are configured to stay on until a technician explicitly turns them off. These likely do not accurately represent the state of the system and aren’t useful for prediction. It is an open question if alarms that are on for a long period of time are still representing anything meaningful and are useful for prediction purposes. The few which do occur a lot tend to occur in bursts, such as Dew Point Fault - ETI Dehydrator ADH-2A COM\(^1\), which occurs very frequently for a few weeks and then doesn’t occur any more. This may be an indication that we have to explicitly model non-stationarity in the dynamics. However, it is also possible the data set clearly represents that the causal factors are no longer present after the alarms stop appearing. Therefore, stationary dynamics would model the phenomenon fine.

Figure 3.4, a CDF for alarm frequency, as with Figures 3.5, 3.6, and 3.9, which are CDFs for the inter-arrival times and alarm durations, had to be done in log scale because there is such a variation in the frequency between alarms.

There are enough alarms in Figure 3.5 that the CDF defines a smooth distribution. Fitting an exponential distribution to this CDF, in Figure 3.7 over-estimates the times between alarms.

Figures 3.7 and 3.8 compare the results of fitting a Weibull\(^2\) versus exponential distribution to the inter-arrival times. The exponential distribution for inter-arrival times rests on the assumption that the data was generated by a Poisson point process. In other words, it implies the placement of the points on the time axis were placed independently of one another. The fact that the Weibull distribution is a better fit suggests that the Poisson point process assumption may fail to capture certain aspects of the data. The Weibull

\[ p(x) = \begin{cases} \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-\left(x/\lambda\right)^k} & x \geq 0 \\ 0 & x < 0 \end{cases} \]

where \(\lambda\) and \(k\) are the scale and shape parameters, respectively.

---

\(^1\)The Dew Point Fault - ETI Dehydrator ADH-2A COM is an alarm that turns on when the humidity in a unit gets too high and the environmental controls must lower it.

\(^2\)The Weibull distribution is
Figure 3.3: Visualization for all the fault log data. White represents a fault, red an alarm, and black is the normal state.
Figure 3.4: Empirical CDF for number of alarm occurrences. This CDF plots the distribution over the number of times a variable transitions from normal to alarm or fault. Each variable is treated as a sample under this distribution.
Figure 3.5: Empirical CDF of inter-arrival times for alarms. The inter-arrival times are defined as the time between the transitions of a particular variable from normal to alarm or fault. The inter-arrival times for all the variables are combined in this CDF. Each transition out of the normal state in the data set is treated as a sample in this distribution.
Figure 3.6: Empirical CDF over mean inter-arrival times for alarms. Alarm inter-arrival times are defined the same here as in Figure 3.5, but here there is one sample for each variable. Each sample is the mean inter-arrival time on a given variable.
Figure 3.7: Fit of inter-arrival time data to exponential distribution. It has a log-likelihood of $-9.4561 \times 10^5$. It has a fitted mean of 49073 seconds or about 13.5 hours. If the data fit an exponential perfectly then it would follow the dashed line. Clearly, the exponential distribution under-estimates the long tail of the data.
Figure 3.8: Fit of inter-arrival time data to Weibull distribution. It has a log-likelihood of $-6.6225 \times 10^5$. The parameters are $a = 1063.2$ (scale) and $b = .3475$ (shape). If the data fit a Weibull perfectly then it would follow the dashed line. Although it fits the data better than the exponential it still under-estimates the activity at the tails. Perhaps a mixture distribution would be best. Given that this is an aggregate of different alarms a mixture model may be appropriate.
Figure 3.9: Empirical CDF of alarm durations. The duration of an alarm is the defined as the time between when there is a transition from the normal state to a non-normal state and when the alarm transitions back to the normal state.
distribution, by contrast, is commonly used in reliability analysis because it represents the time to failure for a system when the failure of its components follow an exponential distribution. The log-likelihood for the Weibull is much higher; the fit can be seen in Figure 3.8. Although they were fitted using a maximum likelihood estimate and the Weibull has an extra parameter for overfitting this can be corrected via use of the BIC to estimate the marginal likelihood. Even after penalizing the Weibull via the BIC the posterior probability of the data being Weibull is nearly one if one operates on an even prior. To be precise the probability of it being exponential is $3.53 \times 10^{-6}$.

The duration follows the same variance in scale as the inter-arrival times. However, it is interesting to note that around 20% of the alarms last around 37 seconds. Some alarms are on a timer. It is possible there is a timer that causes many alarms to turn off around this time. Box plots for the durations and inter-arrival times for the alarms can be found in Appendix A.

### 3.2.2 Similarity Metrics

#### 3.2.2.1 Lag 0 Mutual Information

There is a small subset of alarms that have a large mutual information. Remember the mutual information is

$$I[x, y] = \text{KL}(p(x, y) \| p(x)p(y)) = -\sum_x \sum_y p(x, y) \ln \left( \frac{p(x)p(y)}{p(x,y)} \right)$$

$$I = P_{00} \ln \left( \frac{p_{x=0}p_{y=0}}{P_{00}} \right) + P_{01} \ln \left( \frac{p_{x=0}p_{y=1}}{P_{01}} \right) + P_{10} \ln \left( \frac{p_{x=1}p_{y=0}}{P_{10}} \right) + P_{11} \ln \left( \frac{p_{x=1}p_{y=1}}{P_{11}} \right)$$

Here $p(x, y)$ is bivariate binary distribution over random variables $x$ and $y$. The marginals on $x$ and $y$ are represented by $p(x)$ and $p(y)$. I’ve used a shorthand of $P_{ab}$ for $P(x = a, y = b)$.

If one estimates the joint probability between alarms, treating them as binary by ignoring the difference between an alarm and fault, one can estimate

$$P(x = 1, y = 1) = \int \tilde{W}_x(t)\tilde{W}_y(t)\,dt / D$$  \hspace{1cm} (3.5)

$$P(x = 1) = \int \tilde{W}_x(t)\,dt / D$$  \hspace{1cm} (3.6)

$$\tilde{W}_x(t) = \text{notZero}(W_x(t))$$  \hspace{1cm} (3.7)
$D$ represents the total duration of the data. The probability of $x$ and $y$ being on is just the portion of time they were both on. Likewise, the probability of $x = 1$ is just the portion of time it was on. The other quantities can be calculated using the rules

\[
P(x = 0, y = 1) = P(y = 1) - P(x = 1, y = 1)
\]

\[
P(x = 1, y = 0) = P(x = 1) - P(x = 1, y = 1)
\]

\[
P(x = 0, y = 0) = 1 - P(x = 1, y = 0) - P(x = 0, y = 1) - P(x = 1, y = 1)
\]

for any binary random variables $x$ and $y$. The marginals were easy to extract from the SQL database. The joint was harder and the SQL command to calculate it ran extremely slow. Consequently, I constructed the joint time matrix in MATLAB from the raw data. Inferring the probability of one alarm being on and the other off via the sum to one constraint made things much easier.

A graph showing the links of high mutual information is shown in Figure 3.12. Figures 3.10 and 3.11 show a heatmap of the mutual information between alarms. The alarms form disjoint but nearly fully connected clusters. Looking at the meaning of the alarms many of the clusters seem to be made of alarms that are similar. The system contains disjoint and almost fully connected cliques. The degree to which this occurs is almost surprising. However, an engineered system will be divided into separate subsystems. It makes sense that a failure in one of these subsystems will be related to a failure inside the same subsystem and unrelated to failures inside separate systems. If this is an accurate representation of the system it is a good thing. Because we can model each alarm as being related to a bunch of latent variables. If these subsystems are only weakly related the inference will much more tractable.

3.2.2.2 Lagged Mutual Information

We are interested in predicting one alarm based on the value of the others. Therefore, the mutual information between two variables at the same time is not the most interesting statistic. Computing the mutual information or cross-correlation over different time lags is of more interest. There is a statistical problem with this, however. If an alarm only occurs once on two variables they will have maximal correlation and possibly high mutual information at the time lag between the occurrences even though there is only one occurrence; I refer to this problem as the coincidence problem. There are various workarounds.
Figure 3.10: Heatmap for mutual information matrix. Information here is measured in nats.
Figure 3.11: Close up of sorted mutual information matrix in high mutual information area. Information here is measured in nats.
Figure 3.12: A graph between nodes that have high mutual information (> .35 nats). The term “Latched” has been abbreviated with L. The system contains disjoint and almost fully connected cliques.
Mutual information is not as susceptible to the coincidence problem as cross-correlation. Mutual information measures the reduction in entropy. If the original signal has very little entropy, and another signal predicts it, even perfectly, the reduction in entropy will be small. Therefore, the mutual information will be small. The duration of the alarm could have a great effect on the entropy. If an alarm occurs only once but lasts a minute then the signal will have almost no entropy. If it lasts for weeks then the signal will have more entropy because it is harder to predict the value at a random time point. This means the use of the cleared field will have a big effect on the time lagged mutual information. This is undesirable because in many cases the cleared field is meaningless. The data is reduced to a set of point processes by only looking at the start time.

The point process must be converted to a sequence of random variables. An obvious approach is to divide the waveform into intervals and set an interval to one if it contains a point. This is the alarm edge representation. Picking the appropriate window size will have a large effect. If alarm B usually occurs after Alarm A, but the delay is always a different number of windows after A then the mutual information will never be large. For instance, the delay could sometimes be 5 windows, but at other times 6, 7, or 8, no shift will make the mutual information as large as it should. It isn’t utilizing the fact that these windows are right next door to each other. However, if the window size is expanded windows 5, 6, 7, and 8 will merge together. The mutual information will be larger at this setting. So, the mutual information metric will have to check various window lengths and shifts for all combinations of alarm. This means the mutual information will need to be calculated alarms × alarms × windows × shifts times. The mutual information code has to be highly optimized for this to run in a reasonable time. Using the DataPath data set, it would take several years to compute using a naive implementation for mutual information.

It is possible to tackle this problem if one wants to use cross-correlation. One way is to divide the data set into time windows and compute the cross-correlation within these windows and average out over all the windows. There is a problem here in selecting the right window size. If it is too small it will miss correlations between the data points across window boundaries. It is a somewhat crude metric, but we are not concerned in the exploratory data analysis at getting exact results on the relationships between variables. We only need something that approximately represents the relationships between the variables even if there are some false positives and negatives.

Although the issues with mutual information and correlation affect the non-time lagged version they aren’t as severe because it isn’t able to shift the curves and a dis-
tance that maximizes the correlation that is purely the result of a couple of events. If two events are very closely related without having the flexibility to overfit via shifting it is much more likely to be significant.

It is a somewhat open question what behavior we would like from the similarity measures. For one, there is the issue of selecting the appropriate window function for a time series, which is needed in order to apply the mutual information or correlation metrics. However, there is a deeper question. In the DataPath data set there are about 40,000 time intervals. Suppose that in series A a single alarm happens. Then in series B there is also a single alarm 3 time intervals later. What should we conclude about the relationship?

One way is to approach via a model comparison method. We could start with model A, which assumes the two time series generate points independently and place the points uniformly over time. Then there is model B, which assumes that points in time series B are generated via a Poisson point process where the probability density decays exponentially after a point in series A. Model A will have a huge complexity penalty. The model comparison will strongly favor Model B. Is this desirable behavior given there is only one example in each series?

Suppose you’re a caveman, and you see lightning and then hear thunder a few seconds later. It may be reasonable to assume the two are related. What about the relationship between the massive stock market drop in October 2008 and the impending October 2008 divorce of Madonna. Both events will rarely be observed, but they probably are not related. These judgments are partly related on prior knowledge of the causal nature of the world. And the model comparison results will depend strongly on the prior assumptions given there is only one observed event. It may be possible to justify the caveman’s conclusion by building into the model that there is a rare event in a time series C, which is a strong storm. A strong storm is a novel observation. Then during that period thunder and lightning, other rare events, are also observed together.

3.3 Other Challenges

I am currently working on a single data set. However, there are over 50 potential units where I could receive data. All these units will be somewhat different. However, since many of the alarms will be rare, it may be necessary to estimate some of the parameters using all the data sets in order to gain statistical strength. This type of problem is often called multi-task learning.
Figure 3.13: Graphical model for standard Gaussian mixture model (single task)

Figure 3.14: Graphical model multi-task Gaussian mixture model over $K$ tasks
The difference between multi-task learning and standard learning is illustrated with graphical models in Figures 3.13 and 3.14. The multi-task graphical model represents a data set where there are $K$ domains of learning. In each domain, there are $N$ data points to learn from.

The DataPath problem certainly contains a multi-task aspect across installations. It could even have an additional level of hierarchy. The top level could be product, such as DataPath hardware or some custom hardware used by NBC. The lower level domain would be an individual installation, which may have its own idiosyncrasies.

### 3.4 Baseline Results

Before proceeding to more complex models it is good to establish a base line in performance with basic models. Four basic models are considered: linear regression and logistic regression using either the original data or a low dimensional projection. The low dimensional projection was created with principal components analysis, which is covered in Section 4.2. For the original data the alarm edge representation was used. However, the places where the alarm was on but not at an edge was replaced with NaNs\(^1\) instead of zeros. This can be referred to as the hybrid matrix. It is used because the factors that cause the alarm to go on may be present after the edge for a while. Setting these to a negative case might confuse simple algorithms.

The features, either the hybrid matrix or the low dimensional projection, from one 15 minute interval, are used to predict the value of the hybrid matrix in the next 15 minute interval. Using the alarm level matrix in evaluation would be unfair because predicting the alarm will be whatever value it was last time would be highly accurate but quite useless. The edge matrix might not be good either because if the algorithm predicted no alarm would be on when the alarm is actually on, but not new, is a little unfair. Hence, the hybrid approach is used on the time points where the alarm is on but not new; these points are ignored for training and evaluation. The performance is summarized in ROC and PR curves.

The alarm Dew Point Fault - ETI Dehydrator ADH-2A COM (variable 5)\(^2\) was chosen for performance evaluation because it occurs over a wide time scale and not just a few

---

\(1\)NaN stands for not a number. It is defined in the floating point specification IEEE 754. It is the result of operations that make no sense such as $0 \div 0$.

\(2\)The Dew Point Fault - ETI Dehydrator ADH-2A COM is variable 5 out of 388 in an arbitrary variable indexing scheme.
times. However, the choice was somewhat arbitrary as I could have demonstrated the baseline performance with many other alarms too. The data set was divided so that 67% of the alarms occurred in the training set and the rest were reserved for test. In this example the training set contained 8639 examples of which 137 were positive (1.6%); the test set contained 20644 examples of which 58 were positive (0.3%). The ROC and PR curves for linear regression, linear regression with dimensionality reduction, and logistic regression with dimensionality reduction can be seen in Figures 3.15, 3.16, and 3.17, respectively.

The input to the baseline algorithms was the status of all the alarms at time $t - 1$ and the algorithms were expected to predict the state of alarm 5 at time $t$. This was done for all values of $t$ in the test set. In the feature matrix used, time was discretized to 15 minute intervals. So, the algorithms were basically tested on their ability to use the current state to predict 15 minutes into the future.

It is important to note that the ROC curve is hard to estimate given the small number of samples. However, the results of linear regression of the dimensionality reduced version look encouraging. Logistic regression has the advantage of having a probabilistic interpretation. Linear regression could predict odds above one. Logistic regression does not seem to perform very well. Logistic regression did not even work on the original data matrix because it is too ill-conditioned. The dimensionality reduction helps ensure the matrix is better conditioned.

It is always possible to create a classifier whose ROC curve has the convex hull of a given classifier (Davis and Goadrich, 2006). This would improve these classifiers, especially the logistic regression example. However, the convex hull may naturally fill in a bit if the test set grows. In other words, the observed ROC and PR curves could be partially noise as a result of the small number of positive examples in the test set. The precision never gets above 8%; meaning most of the warnings from such a system would be false. However, if the ROC for linear regression with dimensionality reduction is accurate then most of the failures could be predicted while warning on 10% or so of the non-failure cases.

Despite how much better Figure 3.16 appears it doesn’t dominate the other curves. Logistic regression is better than linear regression at the low false positive end and low recall-high precision area. These two facts are two sides of the same coin as Davis points out that a curve will dominate in ROC space if and only if it does in PR space (Davis and Goadrich, 2006).
Figure 3.15: Performance for linear regression on Dew Point Fault - ETI Dehydrator ADH-2A COM (variable 5). It is interesting that the PR curve goes up and then down instead of a steady downward descent. Usually, as you set the threshold lower the precision becomes lower because the extreme positive predictions become polluted with the moderate ones. Because having a moderate threshold is better it suggests the extreme positive predictions are actually poor indicators of an alarm happening. The moderate ones appear to be a better indicator for this variable. Most likely something changed in the system state between training and test.
Figure 3.16: Performance for linear regression on Dew Point Fault - ETI Dehydrator ADH-2A COM (variable 5) with a dimensionality reduced version of the data
Figure 3.17: Performance for logistic regression on Dew Point Fault - ETI Dehydrator ADH-2A COM (variable 5) with a dimensionality reduced version of the data
3.5 Conclusion

One of the messages of the chapter should be that translating the MaxView log into mathematical form is not trivial. There are also some arbitrary decisions to be made, especially with regard to the feature matrix. We had to make decisions about finely to discretize time. There were decisions about whether it is important to include the end times of the alarms. What paradigm should be used for the feature matrices. Right now, current system is using a 15 minute level of discretization and using primarily the starts of the alarms, or the edge level representation. These settings may not be optimal but one has to make a decision, which may need to changed in retrospect.

There were interesting results when looking at similarity metrics to get an idea or what types of alarms are related. They are also several ways to look at preliminary statistics. The one that has been of primary focus is mutual information. There seems to be a clique nature to alarms that are related. This makes since for a system that has been engineered into subsystems.

The results of the basic methods were explored and some of them, such as linear regression with dimensionality reduction, appears to show that it can provide some prediction. It is unclear if the baseline methods are good enough to provide a useful system.
Chapter 4

Feature Extraction for Time Series

4.1 Temporal Data Analysis

Although the focus of the report is on generative models for prediction in temporal data, other types of analysis are useful to get insight into the data in order to build models. After all, who wouldn’t check the autocorrelation of a univariate time series when trying to do prediction? The focus will be on methods for time series as opposed to sequences because the DataPath data set is made of data at real times rather than just an order.

Preprocessing is an important step in time series analysis. For some applications denoising and outlier removal is an important step. This can be self-defeating when doing predictions on rare events. Trend removal is another common step if the model being used assumes the data is generated in a stationary manner. Sometimes simple transforms make the data much easier to work with. In financial applications, the price data is usually converted into log returns over a given time interval. Before that, it is common to convert it from the space of bid and ask to mid price and spread. Although, this is just a simple linear transform it separates two concepts in the price. In speech applications it is common to first convert the raw speech waveform into a spectrogram because the frequency domain more clearly represents the state of the signal.

Finally, missing values have to be dealt with. If the model can explicitly handle missing data then nothing has to be done here. If it cannot then linear interpolation is a simple solution. The last available sample is sometimes used in time series. For instance, in financial data, there can be a small period of time where there are no trades. This might not be a good approach for hours when the market is not open because it might confuse some models that the price is constant for hours on end.
More fully formed feature extraction algorithms include compressing a multivariate time series to contain fewer series using a dimensionality reduction technique such as principal components analysis, covered in Section 4.2, or independent component analysis. In speech, the formants, the maxima of the spectrogram, at a given point in time are used as features or sufficient statistics in statistical terminology.

4.1.1 Similarity Metrics

Similarity metrics are usually the next step in the exploratory analysis of temporal data. The mutual information analysis done in the data overview section is an example of a similarity metric. The area of similarity metrics is split between ones for numeric time series and symbolic time series. Within numeric time series there are four areas: shape based, feature based, model based, and compression based methods. Shape based methods attempt to capture the qualitative appearance between time series. Feature based methods generally treat the data as if it was iid data. So, statistics like cross-correlation fall in the category of feature based methods. Model based methods will fit a simple model to the data and use the trained model parameters for interpretation. Compression based methods look at how well a time series can be compressed by itself and when paired with another time series. So, some comparison of the entropy of a single time series and the entropy when paired with a new time series, possibly through concatenation, could be considered a compression based method. (Morchen, 2006)

A simple shape based metric is to take the Euclidean distance between two time series. This treats each series as a point in high dimensional space where each time point is a single dimension. This can be generalized to other $L_p$ norms. Shifting and scaling of the time series could fool these metrics. Algorithms will sometimes shift and re-scale time series to correct for this. Shifting and scaling are only a good idea if one is interested in similarity in shape. If you’re interested in how one time series can predict another, arbitrary re-scaling and shifting could overfit and find predictiveness when there is none. This is the same problem as using the cross-correlation with the DataPath data set. Additionally, there are some sanity measures which need to be taken with distance measures. For instance, the scaling factor should be positive, but not too big, which would just normalize noise. Non-uniform sampling and high noise areas are also tricky (Antunes and Oliveira, 2001).

A cousin of the distance measure is the Pearson correlation and the cosine measure. The cosine measure finds the angle between the two time series in high dimensional
space. The cosine measure is equivalent to correlation when time series are standardized such that their means are set to zero. Note that the correlation measure assumes the time points are iid in its probabilistic derivation. The correlation will also have a large confidence interval if the time series are short. The distance metric may be better for short time series (Møller-Levet et al., 2003).

Many of the feature based methods involve looking at descriptive statistics of the data set. For instance, possible methods are the measures of central tendency for the data set: the mean, variance, skew, and kurtosis. The central tendency measures could be used on the first order differences as well. The top coefficients of the discrete Fourier transform (DFT) and discrete wavelet transform (DWT) are also used.

Model based methods involve fitting a standard model such as an ARMA, discussed in Section 5.1, or HMM, in Section 5.3, model to each time series. Model based approaches can be done in two ways. First, they can fit the model to series A and then calculate the likelihood series B was generated by the model fit to series A. The inverse is done next. The average of these two numbers is used as the similarity metric.

\[
\hat{\theta}_A = \arg\max_{\theta} P(A|\theta) \quad (4.1)
\]

\[
\hat{\theta}_B = \arg\max_{\theta} P(B|\theta) \quad (4.2)
\]

\[
S = \frac{P(B|\hat{\theta}_A) + P(A|\hat{\theta}_B)}{2} \quad (4.3)
\]

Ghahramani and Heller (2005) use a slightly different scoring metric in the context of Bayesian sets. A model \( M_1 \) assumes both series come from the same distribution and \( M_2 \) assumes they have different distributions.

\[
S = \log \frac{p(A, B|M_1)}{p(A|M_2)p(B|M_2)}
\]

The similarity metric here is the log odds ratio between the two modeling possibilities. The unknown parameters in each model are integrated over

\[
p(A, B|M_1) = \int p(A|\theta)p(B|\theta)p(\theta)d\theta \quad (4.4)
\]

\[
p(A|M_2) = \int p(A|\theta)p(\theta)d\theta \quad (4.5)
\]

\[
p(B|M_2) = \int p(B|\theta)p(\theta)d\theta \quad (4.6)
\]
Here a full Bayesian treatment is assumed where as in the previous similarity metric a maximum likelihood or MAP estimate is done to get the parameters.

The second approach is to compare the coefficients fit to each model. For instance, in the HMM case the parameters, particularly the transition matrices, for the model fit to each model can be compared. With an HMM method as well as an LDS approach, in Section 5.2, this can be tricky since there are identifiability issues in learning. In other words, there are multiple settings of the parameters which fit all data sets exactly the same. Additional measures must be taken to identify such equivalence classes.

For symbolic time series similarity measures, shape based is replaced by proximity based. The simplest is the Hamming distance, which counts the number of time points where the series have different values.

\[ H(A, B) = \sum_i \text{different}(A_i, B_i) \]

Here \( H \) refers to the Hamming distance. The function different is one if the two arguments have different values and zero otherwise. The time series \( A \) at time \( i \) is represented by \( A_i \). The Levenshtein distance counts the minimum number of insert, delete, and substitutions series \( A \) needs to become equivalent to series \( B \). Token based methods count equivalent sub-series when the entire series are divided using a sliding window.

The mutual information between series \( A \) and series \( B \) can be used as a feature based similarity metric. The mutual information between series \( A \) and series \( B \) is based on the joint distribution between the two series, which can be calculated by looking at co-occurrence rates during the time series. The mutual information over different time lags (as is done with cross-correlation) can be done as well. The cross-correlation between the two series can be used, but mutual information is more elegant for categorical distributions. The correlation will depend on the magnitude of the numerical value assigned to each category. This does not make much sense for a categorical value.

### 4.2 Principal Component Analysis

Principal Component Analysis (PCA) is a useful technique for reducing the dimensionality of data. It was used as a feature extraction technique to get a baseline for performance on the DataPath data set. Performing the regression on the PCA features performed much better than using regression on the original data matrix. Probabilistic
PCA (pPCA) falls in the class of models described by Roweis and Ghahramani (1999) in his collection of linear Gaussian models by setting the parameters such that it severs the temporal connections in the model.

\[ x_n \sim N(0, I) \]
\[ y_n = C x_n + v_t, \quad v_t \sim N(0, R), \quad R = \sigma^2 I \]

The graphical model for pPCA is in Figure 4.1.

![Graphical model for pPCA](image)

Figure 4.1: Graphical model for pPCA. \( y \) refers to the observed data, \( x \) is the low dimensional projection of \( y \). \( C \) is the matrix describing the linear manifold between \( x \) and \( y \). \( R \) is the covariance matrix on the noise for \( y \).

Standard PCA is attained by setting

\[ R = \lim_{\sigma^2 \to 0} \sigma^2 I \]

It is necessary to set the variance of \( R \) to zero in the limit because if the variance \( R \) was exactly zero then the likelihood of any data set that doesn’t exactly fit a linear manifold would be zero. Although it has a relatively simple probabilistic interpretation it was originally derived in a non-probabilistic way. The derivation was to find a new coordinate system for the data. The first axis was selected as the direction which maximized the variance of the data set along the selected direction. The second axis was selected as the one which maximizes the variance among the remaining orthogonal directions. The operation is repeated until a complete coordinate system is created (Bishop, 2007). If one takes the top \( M \) axes the location of the data points will approximately fall on a linear manifold. The axes are computed by

1. Take the covariance matrix, \( S = \text{Cov}(Y) \), of the original coordinates under the data \( Y \).
2. Create a matrix, \( W = \text{Eig}(S) \), of the eigenvectors for \( S \) ordered by their corresponding eigenvalues.

Then low dimensional projection can be found via \( X = WY \), where the columns corresponding to the smaller eigenvalues are removed.

PCA has many interpretations, which include

1. Do inference on the generative model in Figure 4.1
2. Find the directions of maximal variance
3. Minimum error in relative distances in low dimensional projection
4. Find optimal low dimensional linear manifold in RMSE sense
5. Maximize mutual information between the original data points and their low dimensional projection

These properties are very important when interpreting the results of a lower dimensional projection. Besides providing features in an unsupervised way, it is also good for visualizing high dimensional data in two or three dimensions, which we can see. Although the graphical model seems overly simple it can provide interesting results in many cases.

It is also interesting to reproject the low dimensional embedding into high dimensional space. Adding noise in the direction of the disregarded eigenvectors does not change the low dimensional projection. This effect can be utilized to see an “equivalence class” of images.

4.2.1 Eigenface Experiment

PCA was performed on 25 faces, shown in Figure 4.2, to a lower dimensional manifold by Serrano (2004). The principal components, or principal faces, of that manifold is displayed in Figure 4.3.

Images are a rectangular arrangement of pixels. So, they can be represented by matrices. Three matrices are used for color images and one is needed for gray scale. The entire image is reshaped to form a vector when each image is treated as a single data point. Each image becomes a row in the data matrix, which allows a collection of images to be represented in the standard form of a data matrix. It can be used before linear regression or clustering.
Figure 4.2: Original faces used for the eigenface experiment from Serrano (2004)
Figure 4.3: Principal components of the original faces, Figure 4.2, in the eigenface experiment. All of these images are orthogonal to each other when reshaped as a vector.
4.2.2 Random Square Experiment

Here I use an example with synthetic data to illustrate how PCA can be used. Given that the true generative distribution is known the way in which PCA behaves may be better understood through this experiment. In Figure 4.4, a square is located at a random point in space. Then some noise is added to the image. Since it is a binary image this means some pixels are flipped. The task is to find the location of the squares upper left via corner via PCA and then linear regression on the low dimensional embedding.

![Image of a random square experiment](image)

Figure 4.4: A typical sample in the random square experiment

The original image is 20 by 20 pixels. The images were reduced to a subspace of dimensionality four. There were 1000 training examples.

Everything about the state of the square was stored in a four dimensional sufficient statistic. From Figure 4.6 it is clear the model assumes there is a sort of smoothness in the data. None of the hard boundaries of the square are represented. It is assuming a negative correlation between the exterior and the interior. This causes the inside to be black while the squares area is white as seen in Figure 4.6. It doesn’t realize that everywhere outside the square is negatively correlated to what is near by it. However, even if the representation was perfect it would take two dimensions to represent the location of the square. A linear manifold is able to represent the relationship fairly well with 4
Figure 4.5: RMSE values for inferred position by real position. For instance, the color at location (5,5) is the RMSE error between (5,5) and the inferred location by the algorithm when the square is actually located at (5,5). The error is quite low in most cases. Off by just one or two pixels, which could be explained by round off error given that the true location is always an integer. However, the performance is quite bad in the corners. This is to be expected given that for all corner but the upper left most of the square will be off the image. Even in the upper left corner the performance is a bit lower.
Figure 4.6: Reconstruction for random square back of Figure 4.4 into the original 400D space
dimensions even though there was no obvious linear relationship in the construction. Due to the synthetic nature of this data set a hand crafted algorithm would likely do much better, but that is commonly not the case for real data. The error for inferring the square location as shown over various real positions is shown in Figure 4.5.

### 4.2.3 PCA on the DataPath Data Set

So far, I have used many graphics examples to illustrate PCA. It can also be used on time series. For the baseline methods on the DataPath data set I applied PCA to the original figure shown in Figure 3.3.

![PCA reduction of fault time series](image)

Figure 4.7: Features for DataPath time series. All 388 alarms have been reduced down to 25 principal components.

Treating each time point as an observation you can create a low dimensional representation, which some algorithms can work with easier. Conversely, you can treat each
time point as a variable and represent the alarms in a lower dimensional space, as in Figure 4.9, potentially for clustering.

Sometimes lines appear in the low dimensional projection, which can be seen in Figure 4.9. This could potentially be an artifact of the data having a few integer values. The reconstruction is in Figure 4.8.

Sometimes rings and other patterns can emerge in the projection. It is possible this suggests a periodicity in the original data. If one creates a low dimension projection that is a circle then the higher dimensional projection will form a set of sine waves. This is because a circle is a parametric curve defined by a cosine and a sine. A linear transform of these will yield a sinusoid of a particular phase and amplitude. It is possible that other near circular shapes, such as a donut, could suggest that the original data is a particular collection of various sine waves. This implies there is a connection between the low dimensional projection of a time series and its Fourier decomposition.

PCA is similar to other linear Gaussian models such as linear regression in that it is sensitive to outliers. An outlier in the observed space will have a large effect on the projection coefficients. Archambeau et al. (2006) presents a robust form of PCA where the latent data points are assumed to be distributed according to a student $t$. In addition,
Figure 4.9: Projection of the time points to two dimensions. Places where alarm 9 is on are shown in red. Note that the alarm being on in a time interval is not the same as it going from the off state to on in that time interval.
the noise in the observation space is also assumed to come from a student $t$ distribution (Archambeau et al., 2006). This makes it easier for the model to assume a single point may be extreme as a result of noise and not part of a reflection on the latent space.

Also note that the data in my case is mostly binary, which doesn’t fit the Gaussian assumption very well. Exponential family PCA (EPCA) deals with general exponential family distributions, which includes the Bernoulli distribution (Collins et al., 2001). EPCA would be well suited for the fault log data. Some of my data is in an analog form, which may be better suited for standard PCA. Doing both at the same time might require use of a mixed output PCA.

4.3 Clustering

Just as PCA was useful for feature extraction by summarizing the key variances, clustering can also be useful for feature extraction. Clustering divides data sets into categorical groups. In some cases, these groups can be very powerful features to distinguish data points.

Clustering for time series can be divided into three classes: whole series clustering, sub-series clustering, and time point clustering. In whole series clustering, the univariate components in a multivariate time series are clustered into groups. In time point clustering, each time point in possibly multivariate time series is clustered into a group. Sub-series clustering involves clustering chunks of time series into groups. Each chunk possibly represents a different mode or regime of the time series.

For whole series clustering, once a similarity metric has been found it is easy to create a similarity matrix between the time series. At this point any number of standard clustering methods can be applied. If the similarity metric is Euclidean distance then the time series is necessarily embeddable in Euclidean space and a variety of methods that operate directly on the data point can be applied. If the similarity metric is Euclidean distance of the raw data points, then k-means would be an option as a clustering algorithm. Many of the similarity metrics discussed have been applied such as comparing the DCT or DWT coefficients of the time series. PCA can be used to reduce the dimensionality of the vectors if one treats the time series as a set of high dimensional vectors; the eigenvectors in such a form of PCA are known as eigenwaves. (Morchen, 2006)

For time point clustering, any clustering algorithm can be used if one ignores the temporal order of the data. The transpose of the k-means algorithm applied for whole series clustering could be used. In other words, one would treat each time point as a vector
whose elements are given by the values of the time series at that point. Likewise, PCA can be used on these vectors to reduce them to lower dimensional space. This compresses a multivariate time series so that only a few of its univariate components have much magnitude. Additionally, the reconstruction error from the dimensionality reduction can be used as a test for a rare event. In other words, when the inverse operation is done and the point is placed back in high dimensional space, if it is very far from the original point an anomaly alarm can be raised.

Pure model based approaches can be used for time point clustering as well. One approach is to model the time series with an HMM. The hidden states found with the Viterbi algorithm can be used as clusters.

For sub-series clustering a typical approach is to divide each time series up using a window function and then use a whole time series clustering method on the resulting chunks. One may be tempted to use a sliding window for dividing the time series, in order to not chop a pattern up in the middle, but this can produce meaningless results (Lin and Keogh, 2000). Finding the appropriate window length can be tricky. The window size can be set based on the periodicity which can be investigated via an FFT (Hebrail and Hugueney, 2003).

Once the time series is reduced to a lower dimensional space you could apply many standard clustering algorithms to get better features.

Many data sets, such as the DataPath data set, are made of binary vectors. These can be clustered as well. The Gaussianity assumption in the mixture of Gaussians can be generalized to other members of the exponential family. For binary data, this would be a multivariate Bernoulli distribution; this model is known as latent class analysis (Bishop, 2007).

The variance estimates can be overstated when using a mixture of Gaussians on a data set with outliers. The entirety of the mixture of Gaussians can handle outliers by assigning certain components to high variance. Consequently, a possible approach is to have each mixture component made of two sub components; each with the same mean, but one of them has a large variance to catch the outliers. The student $t$ distribution can be viewed as an infinite mixture of Gaussians with the mixing proportions following a gamma distribution. If one takes the limit and sets the number of sub-mixture components to infinity the result is a mixture of $t$ distributions. Peel and McLachlan (2000) extends the common EM algorithm for mixture of Gaussians to efficiently fit the mixture of $t$ distributions using an algorithm known as expectation conditional maximization (ECM). (Peel and McLachlan, 2000)
4.4 Probabilistic Matrix Factorization

Probabilistic matrix factorization (PMF) can be thought of as a probabilistic version of singular value decomposition (SVD), which factorizes a matrix (Salakhutdinov and Mnih, 2008). The factor matrices have priors over them in this case. PMF is to SVD as pPCA is to PCA.

![Graphical model for PMF](image)

Figure 4.10: Graphical model for PMF from Salakhutdinov and Mnih (2008)

Alternatively, one can think of it as bilinear mapping between two categorical variables and a response variable, usually a real number; the graphical model is in Figure 4.10. For instance the two categorical variables could be student number and course number while the response variable could be the score on the final exam. One approach to this problem would be to assign each student and each course a real number, or dummy regressor, and then apply a linear model. It is similar to a generalized linear model for categorical variables except that

1. There is a prior on the dummy regressors
2. There is only an interaction term
3. The dummy regressors can be vector valued

The constraint of only having an interaction term is not too great considering these are all latent variables and they can take on vector values, which means there is likely adequate flexibility in the model to capture effects that a model with a linear term could.
To apply it to time series one could merely apply it to the matrix. However, assuming the time series is reasonably stationary there is nothing special about a given time point and doesn’t count as a meaningful factor.

If the time series is at all periodic the factorization can be used to get factors for the trends. PMF could potentially be used as a test of stationarity.

For the DataPath data set I would like to experiment with a full battery of dimensionality reduction techniques for feature extraction

1. PCA

2. SVD on the original feature matrix, as opposed to the covariance matrix which would be equivalent to PCA.

3. Factor Analysis (FA)

4. PMF

The experiment with PCA has already been done. SVD would effectively smooth the data by removing the smaller principal components. This has been done as well. However, I would also like to test it as a feature matrix for the baseline. It would likely have problems for linear regression as it is low rank. For an advanced baseline I would like to apply EPCA, which would likely fit the fault log data better. The fault log data does not fit the Gaussianity assumption of PCA at all. The lines in the low dimensional projection, Figure 4.9, may be an artifact of this miss fit.

FAM can be shown to be equivalent to pPCA except the output noise is restricted only to be diagonal instead of isotropic as it is with pPCA. The normal generative setup is that there are M latent factors which are related to N observations. Each is related via an unknown linear transform. The generative setup does not make it obvious that the relationship to pPCA is close enough that one merely needs to change the constraint on the output covariance matrix.

PMF could also be done and may yield similar but different results than SVD. For one, it is less likely to overfit given its regularization element.
Chapter 5

Standard Time Series Models

5.1 Autoregressive Models

Autoregressive (AR) models and their variants are perhaps the most common method found in classic time series literature. No treatment of time series would be complete without a mention of AR models. AR\((p)\) models, Figure 5.1, are basically a linear regression of the previous \(p\) values to predict the next value.

\[
y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \epsilon_t, \epsilon_t \sim N(0, \sigma^2)
\]

Figure 5.1: Graphical model for AR(2) model. Each observation is a random variable that is dependent on the last two values in the time series.

If \(y_t\) only directly depends on \(y_{t-1}\), in other words satisfies the Markovian property, then it is an AR(1) model.

If there is a constant linear trend to \(y\) the model will fit better after it is accounted for by

\[
y_t - \mu = \sum_{i=1}^{p} \phi_i (y_{t-i} - \mu) + \epsilon_t, \epsilon_t \sim N(0, \sigma^2)
\]
A moving average (MA) model, Figure 5.2, assumes each observation is a weighted sum of the last $q$ draws from a random variable

$$y_t - \mu = \sum_{i=1}^{q} \psi_i \epsilon_{t-i} + \epsilon_t, \epsilon_t \sim N(0, \sigma^2)$$

Figure 5.2: Graphical model for a MA(2) model. The $y$ variables are black nodes because they are deterministic given the random movements, epsilon.

This describes the MA($q$) process. The AR($p$) and MA($q$) processes can be combined to create an autoregressive moving average model; this is referred to as an ARMA($p, q$) mode, Figure 5.3 (Pole et al., 1994).

$$y_t - \mu = \sum_{i=1}^{p} \phi_i (y_{t-i} - \mu) + \sum_{i=1}^{q} \psi_i \epsilon_{t-i} + \epsilon_t, \epsilon_t \sim N(0, \sigma^2)$$

Figure 5.3: Graphical model for an ARMA(1, 2) model

A key property for ARMA models is stationarity. Most classical analyses of ARMA models only apply when they are stationary. A model is considered stationary if the following property holds

$$p(y_1, y_2, \ldots, y_k) = p(y_{s+1}, y_{s+2}, \ldots, y_{s+k}), \forall k, s \text{ where } k \geq 1, s \geq 0$$

In other words, the joint distribution is invariant to shifts on the axis (Tsay, 2002). There is a weak form of stationarity which requires
\[ E[y_t] = \mu \quad (5.1) \]
\[ \text{Var}[y_t] = \sigma^2 \quad (5.2) \]
\[ \text{Cov}[y_{t-s}, y_t] = \gamma_s \quad (5.3) \]
\[ \forall s, t \text{ where } t > 0, \ s < t \quad (5.4) \]

The weak form only requires the key moments to be invariant to shifts along the axis. It can be shown an AR(1) process is stationary iff

\[ |\phi_1| < 1 \]

More generally, an ARMA(\(p, q\)) model is stationary iff the absolute value of all roots for the following characteristic polynomial are greater than one

\[ 1 - \phi_1 m - \ldots - \phi_p m^p = 0 \]

and the absolute values of all roots for the following characteristic polynomial are less than one

\[ 1 + \psi_1 m + \ldots + \psi_q m^q = 0 \]

One frequently quoted statistic is the autocorrelation. The autocorrelation for a stationary time series at lag \(s\) is

\[ \rho_s = \frac{\gamma_s}{\sigma^2} \]

If one operates on a differenced form of the original time series one gets an autoregressive integrated moving average (ARIMA) model. Each level of differencing is like taking a discrete derivative. If one applied \(r\) levels of differencing and then applies an ARMA(\(p, q\)) model one is really using an ARIMA(\(p, r, q\)) model. Differencing can have some interesting effects. A linear trend is removed by first order differencing. With monthly data, an annual trend can be removed by 12\(^{th}\) order differencing (Pole et al., 1994).

There are three key parts of modeling: inference, learning, and prediction. The inference step is unnecessary with AR, MA, and ARMA models because there are no hidden variables. For prediction, the AR model is trivial.
\[ y_t \sim N\left( \sum_{i=1}^{p} \phi_i y_{t-i}, \sigma^2 \right) \]

For the MA and ARMA models one first has to estimate the previous errors by finding the residual for the previous time steps. Then one can apply the same formula as above. For learning, the AR model is again trivial; being equivalent to linear regression one can merely use the least squares method. The MA and ARMA cases are a bit more complicated because the residuals for one time point is dependent on the residuals for the previous time steps via the prediction property discussed above.

The idea can be generalized to multivariate time series, where the current state vector is a matrix multiply of the last state vector plus a drift term. This results in a multivariate AR(1) model.

\[ y_t = \Phi y_{t-1} + \epsilon_t, \epsilon_t \sim N(0, \sigma^2 I) \]

It can be generalized to an AR\((p)\) model by summing over factors from several matrix multiplies at \(t\) different time lags.

\[ y_t = \sum_{i=1}^{p} \Phi_i y_{t-i} + \epsilon_t, \epsilon_t \sim N(0, \sigma^2 I) \]

Determining the order of an ARMA model is also an issue. The Bayesian Information Criterion (BIC) is sometimes used. The BIC is an approximation to the marginal likelihood to a model when the parameters are fit using a maximum likelihood method. If one can get an estimate for the marginal likelihood then Bayes rule can be applied in order to do model selection. Other methods such as the Akaike Information Criterion (AIC) or cross-validation are sometimes used. One is not necessarily restricted to use the BIC when comparing different orders. It is possible to be fully Bayesian when estimating the parameters and do Bayesian model comparison with the marginal likelihoods. However, most of the literature on ARMA models does utilize latent variable models, which require integration, very much. (Franses and van Dijk, 2000)

One extension to ARMA is ARCH models where the errors are correlated at time points. The error variances themselves follow an AR process
\[
y_t - \mu = \sum_{i=1}^{p} \phi_i (y_{t-i} - \mu) + \sum_{i=1}^{q} \psi_i \epsilon_{t-i} + \epsilon_t, \epsilon_t \sim N(0, \sigma_t^2) \tag{5.5}
\]

\[
\sigma_t^2 = \alpha_0 + \sum_{i=1}^{m} \alpha_m \epsilon_{t-m}^2 \tag{5.6}
\]

ARCH models, Figure 5.4, were invented to model the phenomenon in financial markets where volatility occurs in clusters. In other words, the market will move back and forth between modes of high and low volatility (Tsay, 2002). The effect of large effects followed by more large effects is known as the ARCH effect. Heavy tail behavior is observed in ARCH models. There are also GARCH models where the error variances follow an ARMA process.

\[
y_t - \mu = \sum_{i=1}^{p} \phi_i (y_{t-i} - \mu) + \sum_{i=1}^{q} \psi_i \epsilon_{t-i} + \epsilon_t, \epsilon_t \sim N(0, \sigma_t^2) \tag{5.7}
\]

\[
\sigma_t^2 = \alpha_0 + \sum_{i=1}^{m} \alpha_m \epsilon_{t-m}^2 + \sum_{i=1}^{n} \beta_n \sigma_{t-i}^2 \tag{5.8}
\]

GARCH models, Figure 5.5, require two parameters; so, they are referred to as GARCH\((p, q)\) models. EGARCH carries this process out in log space and TGARCH uses thresholds to switch between constants. In finance, EGARCH is used because it allows for asymmetric effects between large positive and large negatives returns. There are a seemingly countably infinite number of permutations for ARMA models. Other types include IGARCH, GARCH-M, and QGARCH models (Tsay, 2002).
There are examples of even simple nonlinear equations that ARMA models cannot fit. One can extend the autoregressive concept to systems where the observation at time \( t \) is a general function of the previous \( p \) observations. In other words, there is a surface in high dimensional space that defines \( y_t \). A Gaussian process can be used to approximate this surface; remember, the inputs are all observed since they are just the previous \( p \) observations. This approach is used by Quinonero-Candela et al. (2003).

An AR(\( p \)) process is a semi-Markovian model since it depends on more than the immediately previous state. An AR(\( p \)) scalar model can be transformed into a vector AR(1) model by preserving the previous \( p \) values in the extra positions in the vector. Basically, each position in the vector is the value above it at the previous time step. The position at the bottom can be a linear regression of the other values in the vector. Although slightly more complex, the same operation can be done on a multivariate AR(\( p \)) process.

5.2 Linear Dynamical Systems

Linear dynamical systems are the most basic form for a latent variable model of continuous dynamics. As shown below, they can be derived from linear differential equations with Gaussian noise. Extensions are available that extend the idea to nonlinear dynamics. In Section 7.4, a model is proposed that combines ideas from linear dynamical systems and PMF to model a binary matrix such as the fault matrix. The Kalman filter is the algorithm to perform inference on the most general variants in the class of linear Gaussian models described in Roweis and Ghahramani (1999).

\[ x_{t+1} = \lambda x_t (1 - x_t) \]

Weigend and Gershenfeld (1994) presents the example of a logistic map, \( x_{t+1} = \lambda x_t (1 - x_t) \). This is a commonly cited example of a simple nonlinear relation with complex dynamics.
5.2.1 Model Setup

Figure 5.6: Graphical model for the LDS from Ghahramani and Hinton (1998)

\[ x_t = Ax_{t-1} + w_t, \quad w_t \sim N(0, Q) \]  \hspace{1cm} (5.9)

\[ y_t = Cx_t + v_t, \quad v_t \sim N(0, R) \]  \hspace{1cm} (5.10)

The linear dynamical system (LDS), Figure 5.6, has an elegant physical derivation as a system following a linear differential equation with noise in the dynamics, but also noise in the observation. Although, for additional elegance one would use a continuous time model it works well for discretized time.

5.2.2 An Illustrative Example

Imagine a simple spring oscillator; it follows the differential equation

\[ \ddot{x} = -kx \]

where \( x \) is the displacement of the mass on the spring, \( k \) is the spring constant, and we assume the weight has unit mass. This can be put into autonomous form by

\[ \dot{x} = v \]

\[ \dot{v} = -kx \]

This can be discretized by multiplying both sides by \( dt \).

\[ x_t = v_t \cdot dt + x_{t-1} \]

\[ v_t = -kx_t \cdot dt - v_{t-1} \]

This can be put in matrix form

\[ x_t = \begin{bmatrix} 1 & dt \\ -kdt & 1 \end{bmatrix} x_{t-1} \]
I’ve used the notation $x_t$ to refer to a vector containing $x_t$ and $v_t$, meaning it describes the entire state of the system. If we are tracking the state of the spring with any sort of sensor there will be some form of noise in our observation, which we assume is Gaussian. We could also assume there is a little bit of noise in the underlying dynamics as a result of thermal variations or wind affecting the spring. The full model then becomes

$$
\begin{align*}
    x_t &= \begin{bmatrix} 1 -kd_t & dt \\ -kdt & 1 \end{bmatrix} x_{t-1} + w_t \\
    y_t &= Ix_t + v_t \\
    w_t &\sim N(0, \sigma^2_w I), \quad v_t \sim N(0, \sigma^2_v I)
\end{align*}
$$

$y$ is the observed displacement of the weight. The parameters on the noise are $\sigma^2_w$ and $\sigma^2_v$. The problem now fits the form of an LDS.

![Run of Kalman filter on noisy observations from a spring oscillator with external forcing function](image)

**Figure 5.7: Simulation of Kalman filter on an LDS**

Identifiability issues, or degeneracy, exist in the LDS. Without loss of generality, $Q$, the noise in the dynamics, can be set to the identity matrix if $A$ and $C$ are changed accordingly. Likewise, the columns of $A$ and $C$ can be permuted if the state vector is rearranged. One can add an additional constraint, that the columns of $A$ must be ordered according to their norms, which creates a unique solution (Roweis and Ghahramani, 1999). Additionally, if $Q$ is set as the identity matrix then the degeneracy in $A$ and $C$ is
removed. The implications of this are that one cannot interpret the noise in the dynamics if the model is trained via maximum likelihood because there are an infinite number of solutions which have an equivalent likelihood. The solution given by an optimization routine will just be by chance as a result of the initialization. A prior on the parameters must be supplied in order to distinguish the different configurations and interpret the system noise.

It is important to note that this is not the only role of the prior. It can also prevent overfitting into some of the degenerate states that maximum likelihood estimation might enter into. The prior also allows for a Bayesian analysis where one does not use a point estimate of the parameters, but considers multiple cases at once. However, this would require using a variational method or a Monte Carlo method for computational reasons.

5.2.3 Inference

Doing inference in an LDS, in other words Kalman smoothing or filtering, involves only a few matrix operations per time step. The goal of filtering will be two find \( p(x_t|y_{1:t}) \) while for smoothing we try to find \( p(x_t|y_{1:T}) \). The equations for filtering are shown in Algorithm 1. A simulation of the spring oscillator along with the results of the Kalman filtering algorithm are shown in Figure 5.7.

**Algorithm 1 Kalman Filter**

1: for \( t = 1 : T \) do
2: \( \bar{\mu}_t = A\mu_{t-1} + B\mu_t. \)
3: \( \bar{\Sigma}_t = A\Sigma_{t-1}A^T + R. \)
4: \( K_t = \bar{\Sigma}_tC^T(C\bar{\Sigma}_tC^T + Q)^{-1}. \)
5: \( N_t = y_t - C\bar{\mu}_t. \)
6: \( \mu_t = \bar{\mu}_t + K_t N_t. \)
7: \( \Sigma_t = (I - K_tC)\bar{\Sigma}_t. \)
8: end for

Lines 2 and 3 represent the projected state of the system in the next time step, \( \bar{\mu}_t \) and \( \bar{\Sigma}_t \), not given its observation. These incorporate a control input, which is optional in a LDS. It is easy to add, but it makes the LDS more like a continuous input-output HMM than a standard HMM. Line 4 represents the Kalman gain matrix, \( K_t \), which is the degree to which new information influences the current inference. Line 5 is the innovation, \( N_t \), the difference between the actual observation and the expected. Line 6 incorporates these two using the Kalman gain matrix to get a new projection \( \mu_t \). Line 7
updates the uncertainty about the projection, $\Sigma_t$; this also uses the Kalman gain matrix. It is important to note that the uncertainty update does not depend on the observation. The Kalman filter shares this with a typical Gaussian process. This is not surprising since a LDS is a special case of a Gaussian process; all links in an LDS represent jointly normal relationships. (Thrun et al., 2005)

If the observation is missing at a given point just skip line 4 and 5. When modeling continuous dynamics like the spring oscillator it might make sense to have several time steps for each sampling period of the sensor. The larger the time step the larger the deviation from continuous dynamics the discretized model is going to be.

For learning we first need the distribution over the hidden states given all observations. In other words, a prerequisite for learning is smoothing. So, a backward pass is also applied in order to do smoothing. This means we run over the chain backward including sequentially integrating information from the future into the inferred state of system. The previous algorithm, filtering, performed the forward pass, which integrated information from the past. The complete smoothing combines the backward pass and the forward pass.

$$p(x_t|y_{1:T}) = N(x_t|\hat{\mu}_t, \hat{\Sigma}_t)$$

The variables $\hat{\mu}_t$ and $\hat{\Sigma}_t$ represent the estimated state of the system after incorporating future information. The variables $P_t$ and $J_t$ are analogous to $\bar{\Sigma}_t$ and $K_t$ in the forward pass.

Algorithm 2 Backward Pass for Kalman Smoothing

1: for $t = T - 1$ down to 1 do
2: \hspace{0.5cm} $P_t = A\Sigma_t A^T + Q$.
3: \hspace{0.5cm} $J_t = \Sigma_t A^T P_t^{-1}$.
4: \hspace{0.5cm} $\hat{\mu}_t = \mu_t + J_t(\hat{\mu}_{t+1} - A\mu_t)$.
5: \hspace{0.5cm} $\hat{\Sigma}_t = \Sigma_t + J_t(\hat{\Sigma}_{t+1} - P_t)J_t^T$.
6: end for

5.2.4 Learning

To learn the parameters the M-step of the EM algorithm is applied (Bishop, 2007). Taking the derivative of the expected log likelihood gives
\begin{align*}
A^{\text{new}} &= \left( \sum_{t=2}^{T} E[xx_{t-1}^T] \right) \left( \sum_{t=2}^{T} E[xx_{t-1}^T] \right)^{-1} \\
Q^{\text{new}} &= \frac{1}{T-1} \sum_{t=2}^{T} \left( E[xx_{t}^T] - A^{\text{new}} E[xx_{t-1}^T] - E[xx_{t-1}^T] A^{\text{new}} + A^{\text{new}} E[xx_{t-1}^T] (A^{\text{new}})^T \right) \\
C^{\text{new}} &= \left( \sum_{t=1}^{T} y_t E[xx_{t}] \right) \left( \sum_{t=2}^{T} E[xx_{t}] \right)^{-1} \\
R^{\text{new}} &= \frac{1}{T} \sum_{t=1}^{T} \left( y_t y_t^T - C^{\text{new}} E[xx_{t}] y_t^T - y_t E[xx_{t}] C^{\text{new}} + C^{\text{new}} E[xx_{t}] C^{\text{new}} \right)
\end{align*}

One can also get a posterior over the initial hidden state
\begin{align*}
\mu_{0}^{\text{new}} &= E[z_1] \\
Q_{0}^{\text{new}} &= E[z_1 z_1^T] - E[z_1] E[z_1^T]
\end{align*}

The expectations can be calculated by
\begin{align*}
E[x_i] &= \hat{\mu}_t \\
E[xx_{t-1}^T] &= J_{t-1} \hat{\Sigma}_t + \hat{\mu}_t \hat{\mu}_t^T \\
E[xx_{t}^T] &= \hat{\Sigma}_t + \hat{\mu}_t \hat{\mu}_t^T
\end{align*}

\textbf{5.2.5 Extended Kalman Filter}

Many systems in real life are at least a little bit nonlinear. Many approximations have been derived for this case as well. One popular approximation is the locally linear one. The update matrix is derived by linearizing the dynamics around the currently estimated point in space.

If one considers the case of a smaller body orbiting a large body with an external force acting on it we get the following differential equation
\begin{align*}
\ddot{x} &= -\frac{g m_2}{r^3} x + \frac{F_x(t)}{m_1} \\
\ddot{y} &= -\frac{g m_2}{r^3} y + \frac{F_y(t)}{m_1}
\end{align*}

The mass of the moving body is \( m_1 \), while \( m_2 \) is the mass of the stationary body. The distance between them is \( r \) and \( g \) is the gravitational constant. \( F_x \) is the external force in the direction of \( x \) and \( F_y \) is for the \( y \) direction. The total magnitude of the external force

69
is constant, but the direction rotates at a constant rate, which adds a spinning effect to the moving body. One can imagine the orbiting body as a rocket where the thruster is rotating at a uniform rate.

If one puts it in matrix form we get

\[
x_t = \begin{bmatrix} x_t \\ y_t \\ v_{x,t} \\ v_{y,t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ -gm_2 dt/r^3 & 0 & 1 & 0 \\ 0 & -gm_2 dt/r^3 & 0 & 1 \end{bmatrix} x_{t-1} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ dt/m_1 & 0 \\ 0 & dt/m_1 \end{bmatrix} F + w_t
\]

Note that we only observe the top two elements in \( x \). In other words, we only observe the position of the body. We don’t directly observe the velocity of the body. The dynamics matrix depends on \( r \), which is a function of \( x \) and \( y \). This can be linearized by using the extended Kalman filter (EKF) (Thrun et al., 2005). The EKF follows the following algorithm

\begin{algorithm}
\begin{algorithmic}[1]
\State \textbf{for} \( t = 1 : T \) \textbf{do}
\State \( \bar{\mu}_t = g(u_t, \mu_{t-1}) \).
\State \( \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R \).
\State \( K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q)^{-1} \).
\State \( N_t = y_t - h(\bar{\mu}_t) \).
\State \( \mu_t = \bar{\mu}_t + K_t N_t \).
\State \( \Sigma_t = (I - K_t H_t) \bar{\Sigma}_t \).
\EndFor
\end{algorithmic}
\end{algorithm}

The state at time \( t \) is assumed to come from a nonlinear function, \( g \), of the state at time \( t - 1 \) and the external force at time \( t \). The observation is assumed to come from a nonlinear function of the current state, \( h \). The inference algorithm uses these nonlinear functions to extrapolate from the current state in lines 1 and 4, but it linearizes them in the other lines, inference steps, where using the nonlinear function directly is too difficult. To linearize them it takes the Jacobian matrix of \( g \) and \( h \) evaluated at the current state to get the matrices \( G \) and \( H \), respectively. In the example here the observation is a linear function of the true state so we can simply use the \( C \) matrix as we would in a
normal Kalman filter. However, $g$ is nonlinear and we must use the Jacobian $G$. $G$ can be represented as

$$
G_t = \begin{bmatrix}
1 & 0 & dt & 0 \\
0 & 1 & dt & 0 \\
\hat{G}_t & 1 & 0 & 1 \\
\end{bmatrix}
$$

and

$$
\hat{G}_t = \begin{bmatrix}
\frac{gm_2}{r_{t-1}^3} \left( \frac{3y^2_{t-1}}{r_{t-1}^2} - 1 \right) \frac{1}{r_{t-1}^2} dt \\
\frac{3gm_2x_{t-1}y_{t-1}}{r_{t-1}^3} \frac{1}{r_{t-1}^2} dt \\
\frac{3y^2_{t-1}}{r_{t-1}^3} - 1 \frac{1}{r_{t-1}^2} dt \\
\end{bmatrix}
$$

The constant terms are a result of the fact that the only nonlinear relationship is between the current velocity and the position at the last time step. A MATLAB simulation of this system is shown in Figure 5.8 while a zoomed out version is shown in Figure 5.9.

![Figure 5.8: Close up of simulation for orbit example](image)

The external forcing function causes the dynamics to become somewhat chaotic. The body will orbit around the stationary body many times before spiraling out of control for a long distance usually before coming back. Eventually, the body will sometimes escape and diverge off on its own path away from the stationary body, as in Figure 5.10. The EKF is still able to get accurate and small error bounds on the position even in the bizarre dynamics observed near the stationary body. All runs look different. It is also possible to add a feedback control to the external force depending on the position. This causes some odd phenomena in the figures in Appendix B.
5.2.6 Particle Filtering

Although the EKF sometimes gives good results the linearizations can sometimes introduce too many inaccuracies. It is also possible to avoid linearizing by using a sampling approach. This is known as particle filtering (Bishop, 2007), which I will not describe in great detail. Most particle filtering algorithms use importance sampling at each time step for inference of the latent state. However, for increased efficiency samples are re-used from the previous time step by using another filtering algorithm as a proposal distribution, which is commonly a linear Kalman filter. Each proposal filter is known as a particle because they propagate their way through the state space. The basis for the importance sampler the recursive version of Bayes’ rule:

\[
E[f(x_t)] = \int f(x_t)p(x_t|Y_t)dx_t \approx \sum_{l=1}^{L} w_l^{(t)} f(x_l^{(t)})
\]

\[
w_l^{(t)} = \frac{p(y_t|x_l^{(t)})}{\sum_{m=1}^{L} p(y_t|x_l^{(m)})}
\]

Again here \(x_t\) refers to the hidden state and \(Y_t\) refers to the observations. The sum implements the standard importance sampler where the weights are represented by \(w_l^{(t)}\). Sampling the hidden state conditioned on the a whole sequence of observations, even when using a simple proposal distribution such as a LDS, can be computationally burdensome. The idea is to use the samples for the last hidden state to get the next sample.
5.3 Hidden Markov Models

Hidden Markov Models (HMMs) are essentially a discrete version of the LDS. The hidden states are always discrete while the observations can be either discrete or continuous. They can also be viewed as an extension of a standard Markov chain which is more equipped to integrate information over time. When predicting the next observation a standard Markov chain will disregard all information except the current observation. By contrast, an HMM will still use past information because the Markov property only applies to the hidden nodes. In Section 7.1, the noisy-OR model is proposed, which could be viewed as an extension of an HMM. Most temporal data models with discrete latent states can be thought of as an extension or special case of the HMM. The ideas for inference and learning used in HMMs are the basis for the methods used on extensions.

5.3.1 Model Setup

In an HMM there is a single hidden state variable, $x_n$, which evolves according to a transition probability matrix $A$. 
\[ p(x_n|x_{n-1}, A) = \prod_{k=1}^{K} \prod_{j=1}^{K} A^{x_n-1,j \times n_k} \]
\[ p(y_n|x_n, \phi) = \prod_{k=1}^{K} p(y_n|\phi_k)^{x_nk} \]
\[ p(x_1|\pi) = \prod_{k=1}^{K} \pi^{x_1k} \]

The \( x \) here is represented in a one hot encoding\(^1\). The exponents are basically a switch which indicates what value in the product to select. They are much more intuitive when working with log probabilities; the exponents become coefficients which are zero when the value is off.

The probabilities for transitions can be represented more intuitively as

\[ p_n = Ap_{n-1} \]

\( p \) is the distribution over \( x \) at the given time point. If the value of \( x \) is known for the previous time step the \( p \) vector has a one in the position of the true state and zeros everywhere else.

Roweis and Ghahramani (1999) are able to integrate the HMM into the linear Gaussian frame via the use of the winner take all, WTA, function. Under this formulation an HMM is equivalent to a Kalman filter where there is one hidden continuous variable for each possible state. The current state is represented by a one hot encoding, meaning the update matrix creates different transition probabilities for each possible column of the state vector.

\[ x_t = \text{WTA}[Ax_{t-1} + w_t], \quad w_t \sim N(\mu, Q) \]
\[ y_t = Cx_t + v_t, \quad v_t \sim N(0, R) \]

The probability of state \( j \) of \( x \) is equal to the probability mass in the distribution over \( w \) where the \( j^{th} \) component is bigger than all others (Roweis and Ghahramani, 1999). It is also possible to do the reverse and generate a Kalman filter by an HMM where the number of states goes to infinity with a highly constrained transition matrix.

Inference on the HMM can be done via the forward-backward algorithm, which is a special case of message passing. This can be coupled with the EM algorithm to learn

\(^1\)Bishop (2007) uses the term 1-of-\( K \) encoding. One hot comes from digital logic. A bus where only one line is active at a time, signaling a mode, is referred to as using a one hot encoding.
the parameters, which is known as the Baum-Welch algorithm. The forward-backward algorithm can give you the state with the highest marginal likelihood at each time point. This is not the same as the most likely sequence of states (Bishop, 2007).

To find the most likely sequence of states one has to use the Viterbi algorithm. In Kalman filters, the sequence of most likely states is the same as the most likely sequence of states. Intuitively, this is because the dynamics, being Gaussian, have a unimodal and continuous nature. It is very hard to get from a state to one state that is very far away. HMMs have much more flexibility in their dynamics.

5.3.2 Baum-Welch Algorithm

The Baum-Welch Algorithm is very analogous to the EM algorithm for a mixture of Gaussians. Given the likelihood of the systems sequence of hidden states the parameters, such as the transition matrix, can be computed in closed form. Likewise, given the parameters the distribution of hidden states can be calculated in closed form. These will correspond to the E-step and M-step, respectively.

The derivation of the Baum-Welch equations begins with the standard EM algorithm.

Algorithm 4 EM Algorithm

1: while Convergence criteria not met do
2: \[ \theta = \{\mu, \Sigma, \pi, A\} = \text{initialize}(D). \]
3: \[ \text{Calculate } P(X|Y, \theta) \text{ (E-Step)}. \]
4: \[ \theta^{\text{new}} = \arg \max_{\theta} E_{X, \theta^{\text{old}}}[\ln p(X, Y|\theta)] \text{ (M-Step)}. \]
5: end while

Step 1 of Baum-Welch can be done by clustering the outputs. The clusters can be used as estimates on the hidden state. Given the sequence of hidden states one can get an empirical estimate of the transition matrix; this is essentially the M-step. It may be best to average the empirical transition with a prior to avoid initializing in a radical space where certain transitions can never happen. Any matrix entries which begin as zero remain at zero for all EM iterations, which would clearly be bad at initialization.

5.3.2.1 Forward-Backward Algorithm

The E-step consists of the forward-backward algorithm, which can be derived via the sum-product algorithm (message passing) or via manipulating the conditional independence properties to re-derive a recursive version of Bayes rule (Bishop, 2007). The structure of
the HMM is regular enough that re-deriving a recursive version of Bayes rule is not much more difficult than applying the formalized procedure of message passing.

\[ p(x_n|Y) = \gamma(x_n) = \frac{P(Y|x_n)p(x_n)}{p(Y)} = \frac{\alpha(x_n)\beta(x_n)}{p(Y)} \]

The messages, \( \alpha \) and \( \beta \), are defined by

\[ \alpha(x_n) = p(y_1, \ldots, y_n, x_n) \]
\[ \beta(x_n) = p(y_{n+1}, \ldots, y_N|x_n) \]

The messages are propagated by

\[ \alpha(x_n) = p(y_n|x_n) \sum_{x_{n-1}} \alpha(x_{n-1})p(x_n|x_{n-1}) \]
\[ \beta(x_n) = \sum_{x_{n+1}} \beta(x_{n+1})p(y_{n+1}|x_{n+1})p(x_{n+1}|x_n) \]

The messages are initialized by

\[ \alpha(x_1) = p(x_1, y_1) = p(x_1)p(y_1|x_1) = \prod_{k=1}^{K} (\pi_k p(y_1|\psi_k))^{x_{1k}} \]
\[ \beta(x_N) = 1 \]

To get \( P(Y) \) one just has to normalize alpha times beta at any time point; remember \( P(Y) \) is a normalization term. It is easiest to do this operation at the last time point where beta is all ones. So, \( P(Y) \) is merely the sum of the alpha vector at the last time point.

\[ P(Y) = \sum_{x_N} \alpha(x_N) \]

### 5.3.2.2 Parameter Updates

For the M-Step one needs to define

\[ \xi(x_{n-1}, x_n) = p(x_{n-1}, x_n|Y, \theta^{old}) = \frac{\alpha(x_{n-1})p(y_n|x_n)P(x_n|x_{n-1})\beta(x_n)}{P(Y)} \]

This is basically the probability \( x_{n-1} \) was in a given state and then jumped to a given state in \( x_n \). This clearly has a close relationship to the transition matrix. If one maximizes the M-step objective with respect to the transition matrix using Lagrange multipliers that enforces the sum to one constraint along the rows one gets
\[ A_{jk} = \frac{\sum_{n=2}^{N} \xi(x_{n-1,j}, x_{nk})}{\sum_{l=1}^{K} \sum_{n=2}^{N} \xi(x_{n-1,j}, x_{nl})} \]

Likewise, optimizing over the initial value distribution for \( x_1 \) using the sum to one constraint is

\[ \pi_k = \frac{\gamma(x_{1k})}{\sum_{j=1}^{K} \gamma(x_{1j})} \]

Clearly, this is just the standard empirical distribution over \( x_1 \). The sum in the denominator normalizes so we can just use the gamma values not the prenormalization probabilities. The mean and covariance of the outputs in the case of Gaussian outputs are

\[ \mu_k = \frac{\sum_{n=1}^{N} \gamma(x_{nk})y_n}{\sum_{n=1}^{N} \gamma(x_{nk})} \]

\[ \Sigma_k = \frac{\sum_{n=1}^{N} \gamma(x_{nk})(y_n - \mu_k)(y_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(x_{nk})} \]

These are exactly the same as for the mixture of Gaussians case; although, the responsibilities come from a different place in that they include sequential information.

### 5.3.2.3 Numerical Difficulties

In an actual implementation numerical difficulties become significant especially when the outputs are continuous. Firstly, alpha and beta can become exponentially smaller every iteration (Bishop, 2007). This is because they constitute a joint probability with the observation space. The probability function must normalize to one over the entire observation space, which grows exponentially with observations. Therefore, most points in the observation space get exponentially smaller in magnitude as that space grows.

The solution is to introduce scaling factors to alpha and beta so that

\[ \gamma(x_n) = \hat{\alpha}(x_n) \hat{\beta}(x_n) \]

We can simply find a factor to normalize alpha and apply it to the beta in the last time step

\[ c_n \hat{\alpha}(x_n) = p(y_n|x_n) \sum_{x_{n-1}} \hat{\alpha}(x_{n-1})p(x_n|x_{n-1}) \]

\[ c_{n+1} \hat{\beta}(x_n) = \sum_{x_{n+1}} \hat{\beta}(x_{n+1})p(y_{n+1}|x_{n+1})p(x_{n+1}|x_n) \]
One can also find
\[
\xi(x_{n-1}, x_n) = \hat{\alpha}(c_n x_{n-1}) p(y_n | x_n) P(x_n | x_{n-1}) \hat{\beta}(x_n)
\]
\[
P(Y) = \prod_{n=1}^{N} c_n
\]

Normalizing alpha after even one step can sometimes be a problem. If the EM algorithm sets the emission parameters in such a way that the probability over a given observation is very close to zero for all of the possible states then alpha will evaluate as all NaNs if implemented naively.

\[
\hat{\alpha}(x_n) = \frac{p(y_n | x_n) \sum_{x_{n-1}} \hat{\alpha}(x_{n-1}) p(x_n | x_{n-1})}{\sum_{k=1}^{K} p(y_n | x_n = k) \sum_{x_{n-1}} \hat{\alpha}(x_{n-1}) p(x_n = k | x_{n-1})}
\]
\[
\hat{\beta}(x_n) = \frac{\sum_{x_{n+1}} \hat{\beta}(x_{n+1}) p(y_{n+1} | x_{n+1}) p(x_{n+1} | x_n)}{\sum_{k=1}^{K} p(y_{n+1} | x_{n+1} = k) \sum_{x_n} \hat{\alpha}(x_n) p(x_{n+1} = k | x_n)}
\]

The division of the emission likelihoods can be done with the division of Gaussians trick. This allows both likelihoods to be very small without causing numerical problems.

5.4 Rare Events

ARMA and LDS models have inherent difficulty dealing with rare events due to their completely Gaussian dynamics. They are sensitive to heavy tailed noise since their output is Gaussian, which means a radical output will make a radical inference onto the state of the system. The internal state of the system will also be resistant to moving very far from the last state. If an extreme event does occur the model will split the explanation between a large noise event and a large move in the hidden state. It will never be able to predict a radical change or even consider it at all likely because it treats everything as one Gaussian cloud.

The switching Kalman filter (SKF) can approximate more complicated dynamics by assuming there are multiple regimes the system can be in Ghahramani and Hinton (1998). Each of the regimes follows an LDS. Likewise, the EKF and particle filtering can be used
to approximate fully nonlinear dynamics. An assumed density filter (ADF) matches the moments between a Gaussian approximation for the distribution at a given time point and what would result if full nonlinear filtering were done. In other words, there is a normal distribution over state $x_{n-1}$ and after a single time step of nonlinear dynamics there will be a weird, and usually intractable, distribution over $x_n$. This distribution is approximated via moment matching. Once a Gaussian approximation for $x_n$ is attained the operation is repeated for $x_{n+1}$.

The HMM offers more promise as it can have a state which has low probability of being transitioned to and has radical output. Mixture of Gaussians output for HMMs gives it more flexibility since it can also deal with heavy tail noise. Some problems however are not particularly suited for an HMM given its fixed state space, such as the spring oscillator.

It would be interesting to compare performance on rare event data sets for all these standard methods: ARMA, LDS, and HMMs. The DataPath data set could serve this purpose. However, the DataPath data set is not the canonical rare event data set as there are many other aspects.
Chapter 6

Extensions to Classic Time Series Models

6.1 Switching Kalman Filter

A switching Kalman filter (SKF) is a linear dynamical system that can switch between different modes of linear operation (Ghahramani and Hinton, 1998). In a simple case, only the output matrix is affected by the state; this is illustrated in the graphical model in Figure 6.1.

\[
x_t = Ax_{t-1} + w_t, \quad w_t \sim N(0, Q)
\]

\[
p(s_t|\pi) = \prod_{k=1}^{K} \pi_{k}^{s_{tk}}
\]

\[
y_t = C^{s_t} x_t + v_t, \quad v_t \sim N(0, R)
\]

Figure 6.1: Graphical model for a simple SKF from Ghahramani and Hinton (1998). The continuous valued hidden state is represented by \(x_t\). The observations are \(y_t\) and \(s_t\) represents the output mode of the system.

The notation for the parameters here is the same as with the standard LDS, but $C$ is index by $C^s_t$ because there is a different $C$ matrix for every value of $s$. One can make a more complex version where the dynamics also depend on the state. At this level of sophistication, approximating nonlinear dynamics, it doesn’t make much sense unless there is some temporal correlation along the switching state. So, we have a Markov chain for the hidden state as in Figure 6.2.

Figure 6.2: Graphical model for more complex SKF from Ghahramani and Hinton (1998)

$$x_t = A^{s_t-1}x_{t-1} + w_t, \ w_t \sim N(0, Q^{s_t-1})$$

$$p(s_t|s_{t-1}, A) = \prod_{k=1}^{K} \prod_{j=1}^{K} A^{s_{t-1}, s_{tk}}$$

$$p(s_t|\pi) = \prod_{k=1}^{K} \pi_{s_{tk}}$$

$$y_t = C^{s_t}x_t + v_t, \ v_t \sim N(0, R)$$

The presence of both a discrete Markov chain and multiple LDS is why these models are known as hybrid models. They are models where the hidden state is mixture between continuous and discrete variables.

It is also possible to add control inputs, which affect the continuous dynamics and observations like in the normal LDS, Figure 6.3. Another extension is to combine an SKF with an autoregressive process over the observation states as in Figure 6.4.

Ghahramani and Hinton (1998) create an elegant extension to the first version, Figure 6.5, where the model observations are a selection from different underlying LDS. The selecting state follows a Markov chain, which allows for strong temporal correlation on what underlying chain is observed. If the current state were generated iid then there would no longer be significant periods of time where a single chain is being observed. The observations would switch back and forth erratically between the possibilities and hence be more limited as a model.
Figure 6.3: Graphical model for SKF with external inputs from Ghahramani and Hinton (1998)

Figure 6.4: Graphical model for SKF with autoregressive outputs from Ghahramani and Hinton (1998)

\[ x_t^i = Ax_{t-1}^i + w_t^i, \quad w_t^i \sim N(0, Q_i), \quad i = 1, \ldots, K \]

\[ p(s_t|s_{t-1}, A) = \prod_{k=1}^K \prod_{j=1}^K A^{s_{t-1}, j, s_{t}^k} \]

\[ p(s_1|\pi) = \prod_{k=1}^K \pi_{s_1}^{s_{1k}} \]

\[ y_t = C^{s_t} x_t^i + v_t, \quad v_t \sim N(0, R) \]

The model can also be seen as a generalized mixture of experts, where each LDS is an expert and the inference model is trying to figure out which one to trust with the current observations. Or equivalently, trying to infer the true switching state. A block diagram representing this view is shown in Figure 6.6.

A version of the EM algorithm is available for learning the parameters of an SKF. The E-step itself is divided into multiple steps one of which involves running Kalman smoothing on each LDS as well as running forward-backward on the state chain. After the responsibilities for each chain have been computed, the M-step can re-estimate the parameters for each LDS. Baum-Welch style estimation can be done on the parameters
Ghahramani and Hinton (1998) presents a variational approximation involving deterministic annealing for learning the parameters of an SKF.

6.2 Novelty Detection with the SKF

Williams et al. (2006) presents a way to detect novel states with an SKF. In other words, states that the model has had little or no previous exposure. His model is based on a SKF where state $s = 1$ is known as the normal state. States $s = 2 \ldots K$ represent known abnormal states. An additional X-factor state is meant to model unknown states. In Williams’ version of the SKF
The state variable will follow some version of a Markov chain itself. The parameters for the X-factor are specified as

\[
x_t \sim N(A^{(s_t)x_{t-1}}, Q^{(s_t)}), \quad y_t \sim N(C^{(s_t)x_t}, R^{(s_t)})
\]

Empirically, Williams found $\xi = 2$ to be a reasonable setting. The X-factor follows the same dynamics as the normal state except for that it has a much larger variance on the observation. It has the highest posterior probability in areas of low probability mass. Effectively, it adds a “none of the above” option when inferring the hidden state. To illustrate his point he uses a 1D and independent SKF with an X-factor.

![Figure 6.7: Distribution over the observation. Each Gaussian represents the distribution on the output conditioned on the system being in a certain state, $s$. The dashed line represents the output given it is in the X-factor state.](image)

The X-factor is centered on the normal state. The other abnormal states are more highly peaked and at the extremes. Note that areas in between the normal state and the observed state could be labeled as belonging to the X-factor, not just points at the extremes. This is illustrated in the probability distributions in Figure 6.7.

In his application he applies the X-factor to a much more complicated system described by a Factorial SKF (FSKF). The graphical model is shown in Figure 6.8.

The hidden state is a cross product of the factors. In other words, each combination of the hidden factors forms a switching state. The artifactual state depends on the true dynamics and the switching state. Furthermore, the observations are the artifactual state with some measurement noise. The artifactual state could be the result of a number of known abnormal states or the X-factor.

He uses this model in a medical application where the true state is the actual state of the patient. The artifactual state is going to be affected by the patient’s state and...
Figure 6.8: Graphical model for an FSKF used on condition monitoring in a medical application (Williams et al., 2006). It illustrates how an X-factor can be used in practice. The state of the sensors, which are the factors. The data observed has some additional observation noise.

6.3 Factorial Hidden Markov Model

The Factorial Hidden Markov Model (FHMM) is an extension to the HMM where there are $M$ hidden states evolving in parallel, shown in Figure 6.9.

Figure 6.9: Graphical model for an FHMM from Ghahramani and Jordan (1997)

In principle there is nothing different about an FHMM from an HMM with a larger transition matrix that is more constrained. If there are two latent chains with $N$ states for the first variable and $K$ for the second then there are $N \times K$ possible states of the system. However, the transitions in the two chains are independent meaning the there are far fewer degrees of freedom in the $N \times K \times N \times K$ transition matrix than usual.

$$ T = T_1 \otimes T_2 \otimes \ldots \otimes T_M $$
The above formula constructs the “mega transition” matrix of the flattened HMM via the Kronecker product, which is represented by the $\otimes$. This means the matrix $T$ is a block matrix with one block for each entry in $T_1$. Each block is made by multiplying that value of $T_1$ with all of $T_2$. $T_1$, $T_2$, and $T_M$ represent the transition matrices for the first, second, and $M^{th}$ hidden chain, respectively. For simplicity imagine only two chains for the time being, the first chain consisting of states 1, 2, and 3. The second chain could consist of $a$, $b$, $c$. Since the chains are independent

$$P((a, 1) \rightarrow (b, 2)) = P(a \rightarrow b)P(1 \rightarrow 2)$$

Therefore, the mega transition matrix is the Cartesian product of the transition probabilities. Hence, the formula above. If there are $M$ hidden chains, one can reduce two of the chains into one via the Kronecker product. This combined chain can be further combined with the remaining chains via the Kronecker product. Therefore, the mega transition matrix of a FHMM with $M$ latent chains is the Kronecker product of all the latent chains’ transition matrices.

Although inference on a FHMM is difficult it is better than working with a much larger transition matrix and the complexity penalty it implements is much better in some cases than the much looser version. A standard HMM will have $N^2 - N$ degrees of freedom for the transition matrix and $N$ degrees of freedom to fit the distribution on the initial state. So, in total, there are $N^2$ degrees of freedom for the hidden dynamics. The additional parameters for the emission probabilities will depend on what type of output is being used. So, in the case of the factorial HMM there are $N^2 + K^2$ degrees of freedom. Whereas in the flattened HMM there are $N^2 \times K^2$ degrees of freedom. The effect of the complexity penalty in favor of the factorial HMM is huge.

### 6.4 Infinite Hidden Markov Model

In a standard HMM the number of components $K$ has to be set in advance, which is not easy to do. One approach is to try various values and see which works best or has the highest marginal likelihood. The infinite HMM (iHMM) assumes an infinite number of latent states and the algorithm infers which finite subset has been observed. The allocation of states in the transition matrix is derived via the Hierarchical Dirichlet Process (HDP). \cite{Gael2008}

There is an analogous problem to determining the number of components in a mixture of Gaussians model. The Dirichlet process (DP) can be used for clustering when the
number of components is assumed to be infinite and again only a finite subset has been observed.

The iHMM is significant in that it provides a non-parametric way to infer latent states over time, which will be shared with a model in the next section.

6.5 Phase Type Distributions

An observation to make about HMM is that the amount of time a chain spends in a given state will have a geometric distribution

\[ P(n) = (1 - p)p^n \]

\( P(n) \) is the probability it stays in the state for exactly \( n \) more time steps. This follows a geometric distribution. The probability of self-transition is \( p \). The geometric wait time can be unattractive if the HMM matches an underlying process that doesn’t have this property. However, one state in the real world doesn’t necessarily match one state in the HMM.

If one considers a compound state with two independent sub states then one can get a mixture of exponentials or hyper-exponential wait time. This allows more probability mass at the extremes, but it doesn’t allow for bi-modal or like behavior. If one allows the two sub-states to jump between each other we can get a larger class of distributions. These are the phase type distributions. (Osogami, 2005)

The full class of phase type distributions offers a great deal of flexibility as it has \( S \times S \), where \( S \) is the number of compound states, parameters for a one dimensional distribution. The structure of the compound state can be constrained, which leads to distributions such as the Erlang, Coxian, and generalized Erlang (Osogami, 2005).
Chapter 7

Research Ideas

The highest priority research idea is a model I call the latent cause model, which is designed for conditioning monitoring problems as seen in the DataPath data set. I suspect it is the most directly applicable to the DataPath problem. It comes in the three forms: autoregressive, latent variable, and non-parametric. The first form assumes that past alarms cause the alarm at a given time step. The second one assumes there is a latent process, which generates the alarms. The non-parametric version will take the limit in the number of possible latent causes to infinity.

The ideas behind an LDS and PMF are combined to create a binary LDS model. Each time series in a multivariate time series will have a certain susceptibility to the underlying state at each time point. Another model, PC-Kalman, is proposed that, in a sense, does automatic feature selection for LDS models. It combines the process of using PCA for feature extraction and an LDS for modeling into one model.

7.1 Latent Cause Model

7.1.1 Autoregressive Causes

In this model, alarm $i$ at time $t$ is a noisy OR of the alarms at the time $t - 1$, but it is only affected by certain alarms. Its possible causes are specified in matrix $X$. It could go on for no reason with probability $(1 - q)$. This idea is illustrated via a block diagram borrowed from digital logic in Figure 7.1.
\[ A_{i,t} \sim \text{noisyOR}(A_{t-1}, X_{ij}, q) \quad (7.1) \]

\[
\text{noisyOR}(A_{i,t}|A_{t-1}, X, q) = \begin{cases} 
1 - q \prod_k (1 - A_{k,t-1})^{X_{ik}} & \text{if } A_{i,t} = 1 \\
q \prod_k (1 - A_{k,t-1})^{X_{ik}} & \text{if } A_{i,t} = 0 
\end{cases} \quad (7.2)
\]

\[ q \sim \text{Beta}(a, b) \quad (7.3) \]

\[ X_{ij} \sim \text{Bernoulli}(y) \quad (7.4) \]

\[ y \sim \text{Beta}(1, 1) \quad (7.5) \]

A simple version can have a single noise level \( q \), while a more flexible version can have a \( q \) vector with one noise level for each alarm. The matrix \( X \) is a binary matrix presenting the causal links. If \( X_{ik} = 1 \) then when cause \( k \) occurs alarm \( i \) will be caused to go on. The variable \( y \) is the prior probability that a causal link exist. There is a beta hyper-prior on \( y \). To increase the amount of information it can utilize while avoiding the use of hidden states it can be extended to use the last \( p \) states as possible causes for an alarm. This is very analogous to the extension of AR(1) models to AR(\( p \)).

![Figure 7.1: Representation of noisy OR process for alarms](image)

Alarms are assumed to be caused by any number of factors or just be a random alarm for no reason. Here the alarm is represented as being generated by an OR gate.

For the simplest version of learning one can do maximum likelihood estimation or MAP estimation on \( X \) and \( q \). \( X \) must be done with some version of combinatorial gradient ascent such as hill climbing. At each value of \( X \), \( q \) must be updated to its maximum value to fairly evaluate the likelihood at a given setting of \( X \). If the cause for an alarm is on then the noise level, \( q \), has no effect on the likelihood. Likewise, if the alarm is on for no reason, meaning no cause was on, then the probability of it being on is \((1 - q)\). Therefore, we can treat it as a coin flipping problem.

There are three types of observations for an alarm at a given point in time: alarm off (A), alarm on for no known reason (B), and alarm on because a cause was on (C). The maximum likelihood estimate of \( q \) is \( B/(A+B) \). The number of time C occurs does
not influence the estimate of the noise level. If the cause occurred then the alarm was
guaranteed to go on regardless of the noise, meaning its presence provides no information
about the noise level. Consequently, once we filter out the observations when the cause
is on it is like a coin flipping problem; we can do a full Bayesian analysis by putting a
beta prior on $q$. For the vector version of $q$ we will need a hyper-prior so that individual
$q$’s can gain statistical strength from each other. This is very important given that each
alarm does not have many positive examples to learn its $q$ from.

The model is compatible with rare event prediction since there is nothing but the
complexity penalty on $X$ stopping the algorithm from linking a rare event to a rare cause
(assuming the phenomenon has been observed, of course).

7.1.2 Hidden Causes

One can instead assume that the causes are actually latent and the mapping is between
a latent state and the $X$ matrix goes to a matrix of latent causes. This is like going from
a Markov chain to an HMM.

EM can be used to learn and do inference in this model. However, both the E-step
and the M-step have computational difficulties. Optimization of $X$ even given the hidden
states is hard given the latent variables due to the combinatorial nature of $X$.

Presumably, the E-step could be performed via (loopy) belief propagation. The belief
propagation algorithm could be simplified given that the nature of the noisy OR distri-
bution. If any of the effects are not on then the cause will certainly be off. So size of the
belief propagation will be reduced. In fact, it could be quite small considering that at
any given time most alarms are off. The relationships between the possible causes and
the alarms could be singly connected in some cases.

In the M-step we have to estimate the causal connection matrix, $X$, and the noise
vector $q$. The coin flipping equation can no longer be used to estimate $q$ because the
causes are no longer observed. A weighted coin flip estimate could be used given the
probabilities of the causes from the E-Step.

7.1.3 Non-Parametric Version: Infinite Causes

If we don’t want to set the number of causes we could do the same thing that is done
in HMMs when you don’t want to specify the number of hidden states. Assume it is
infinite and use a iHMM. Here we could assume there are an infinite number of potential
causes. The way this could be done is to assume the latent cause matrix is sampled from a Markov Indian Buffet Process (mIBP). This is like going from a HMM to an iHMM.

### 7.2 Binary Probabilistic Matrix Factorization

A binary PMF (BPMF) is like a PMF except that the observations are binary and there may be more than one observation per combination of the indices. So, in a sense it factorizes two matrices. One counts the number of positive examples and one counts the number of negative examples.

For example, suppose each row corresponds to a member of the Cambridge ski team and each column represents a member of the Oxford ski team. A positive example is when the Cambridge skier wins a race between them and a negative example is when the Oxford skier wins. A certain combination might never occur. Some combinations might occur more than once. We want to infer the skill level on the team of each skier and predict the probability a Cambridge skier will win for each possible match up. The BPMF could be used for this purpose.

\[
Y = \begin{bmatrix}
5 & 0 & 2 \\
1 & 1 & 1 \\
2 & 0 & 7
\end{bmatrix}
\quad
N = \begin{bmatrix}
0 & 0 & 3 \\
2 & 0 & 1 \\
1 & 0 & 2
\end{bmatrix}
\Rightarrow
P = \begin{bmatrix}
1 & .4 \\
.33 & 1 & .5 \\
.66 & .78
\end{bmatrix}
\]

The \(Y\) matrix counts the number of positive examples and the \(N\) matrix counts the number of negative examples. The \(P\) matrix finds the portion in each category that is positive. As in this example, the \(P\) matrix can contain missing values. This design leads to the graphical model in Figure 7.2.

\[
U_i \sim N(0, \sigma_U^2 I) \quad (7.6)
\]

\[
V_j \sim N(0, \sigma_V^2 I) \quad (7.7)
\]

\[
X|i,j \sim \text{Bernoulli}(g(U_i^T V_j)) \quad (7.8)
\]

Figure 7.2: Graphical model for BPMF
$U$ is a $M \times K$ matrix and $V$ is $N \times K$ matrix. $X$ is a given draw and $i$ and $j$ are its factors which determine were a one will be added in the $Y$ or $N$ matrix. The function $g$ represents a logistic function\(^1\).

\[
P(Y, N|U, V) = \prod_i \prod_j g(U_i^T V_j)^{Y_{ij}} (1 - g(U_i^T V_j))^{N_{ij}}
\]

\[
P(U, V|Y, N) \propto \prod_i N(U_i|0, \sigma^2_U I) \prod_j N(V_j|0, \sigma^2_V I) \prod_i \prod_j g(U_i^T V_j)^{Y_{ij}} (1 - g(U_i^T V_j))^{N_{ij}}
\]

\[
\ln(P(U, V|Y, N)) = \sum_i \sum_j (Y_{ij} \ln g(U_i^T V_j) + N_{ij} (1 - \ln g(U_i^T V_j))) + \frac{1}{2\sigma^2_U} \|U\|^2 + \frac{1}{2\sigma^2_V} \|V\|^2 + C
\]

Taking the gradients with respect to $U$ and $V$ to get a MAP estimate gives

\[
\frac{\partial E}{\partial U_j} = \sum_i (Q_{ij} Y_{ij} - P_{ij} N_{ij}) V_i - \frac{U_j}{\sigma^2_U}
\]

\[
\frac{\partial E}{\partial V_i} = \sum_j (Q_{ij} Y_{ij} - P_{ij} N_{ij}) U_j - \frac{V_i}{\sigma^2_V}
\]

\[P_{ij} = g(U_i^T V_j) Q_{ij} = 1 - P_{ij}\]

Using the properties

\[
\frac{d \ln g(x)}{dx} = 1 - g(x)
\]

\[
\frac{d \ln(1 - g(x))}{dx} = -g(x)
\]

the gradients can be used for an optimization routine. The summation drives $U$ in the direction of $V$ if there are positive examples unexplained and vice versa if there are negative examples unexplained. The subtraction at the end acts as a complexity penalty, which drives $U$ positive if it is negative and negatively if it is positive. This can be simplified further as

\[
\frac{\partial E}{\partial U} = VW^T - U \quad (7.9)
\]

\[
\frac{\partial E}{\partial V} = UW - V \quad (7.10)
\]

\[W = Q \odot Y - P \odot N \quad (7.11)\]

\(^1\)The logistic function is commonly used to squash the real line to a number 0 and 1. It is frequently used when one wants to interpret the output of their model as a probability. The definition of the logistic function is $g(t) = \frac{1}{1+e^{-t}}$. 

93
\( W \) can be thought of as a weight matrix, which is created using the Hadamard product, \( \odot \).

### 7.3 PC-Kalman

Principal component analysis is frequently used to get features for another algorithm. If one tries to use PCA to get the features for a Kalman filter one is essentially creating a restricted output matrix. I used this method in the Machine Learning for Signal Processing (MLSP) competition in a financial application. The observed state in a Kalman filter is a matrix multiply of the hidden state. Here we are using the observed state in the Kalman filter as the hidden state in PCA, which uses another matrix multiply to get the observed data. Therefore, we are really forcing the Kalman filter to have an output matrix that factorizes. Alternatively, we could estimate the two factors jointly. This means temporal correlations will influence the way in which the model picks the matrix that describes the linear manifold. The graphical model for this is the same as for an ordinary Kalman filter, but the equations are subtly different.

\[
\begin{align*}
  x_t &= A x_{t-1} + w_t, \quad w_t \sim N(0, Q) \\
  y_t &= C x_t + v_t, \quad v_t \sim N(0, R) \\
  z_t &\sim \lim_{\sigma^2 \to 0} N(W y_t, \sigma^2 I) \\
  z_t &\sim \lim_{\sigma^2 \to 0} N(W C x_t, W R W^T + \sigma^2 I) = N(W C x_t, W R W^T) \\
  &\text{if } R = I \quad z_t \sim N(W C x_t, WW^T) = N(T x_t, \Sigma) \\
  \text{where rank}(\Sigma) &= p
\end{align*}
\]

The size of \( C \) is \( p \times k \), the size of \( R \) is \( p \times p \), and the size of \( W \) is \( n \times p \). The observed state is \( z_t \) while the hidden state is \( x_t \); \( y_t \) serves as an intermediary in the dimensionality reduction. By combining the feature extraction step of PCA in the LDS the model will be better at propagating the uncertainty of the low dimension projection into the temporal domain. In other words, PCA just gives a point estimate which when fed into an LDS treats it as certainty. We can lessen some of that naivety by combining the two. Likewise, when estimating the parameters the temporal relations help PCA determine the projection matrix.
7.4 Binary Linear Dynamical System

The Kalman filter was invented to model physical processes especially in signal processing. Therefore, its outputs are continuous. The HMM can have continuous or discrete outputs. It is possible to combine PMF and Kalman filters to produce a system with continuous dynamics, but with binary vectors as the observations. Each time series has vector $S_i$ and each time point has vector $Y_t$. There is a matrix, $F$, where $F(i, t) = 1$ if time series $i$ has an alarm go on in time chunk $t$.

\[
x_t = Ax_{t-1} + w_t, \quad w_t \sim N(0, Q) \tag{7.12}
\]
\[
y_t = Cx_t + v_t, \quad v_t \sim N(0, R) \tag{7.13}
\]
\[
S_i \sim N(0, \sigma^2 I) \tag{7.14}
\]
\[
F_{i,t} \sim \text{Bernoulli}(g(S_i^T Y_t)) \tag{7.15}
\]

The $S_i$ vector is analogous to PMF. The $x_t$ variable represents a latent state of the system. On the other hand one can interpret $y_t$ as a noisy version of $x_t$. The vector $S_i$ can be thought of as the susceptibility of alarm $i$ to each of the factors in $y_t$. If one wanted to simplify the model they could set $y_t = x_t$. Like in the Netflix\(^1\) problem, each user has a certain susceptibility to liking a given movie each alarm has a susceptibility to an underlying system state.

7.5 Mixed Output Principal Components Analysis

PCA was originally designed for continuous data. Exponential family PCA (EPCA) extended this to all data which could be generated by the exponential family which includes binary data. But in the DataPath data the space is mixed. There are some binary alarms as well as continuous readings from the analog sensors. EPCA in principle allows for this but hasn’t been applied this way.

This idea can be extended to HMMs to allow them to support multiple data types as outputs.

\(^1\)The Netflix problem is a challenge for $1$ million, which will be rewarded to the first team to improve the Netflix recommender system by 10%. An applicants’ algorithm must predict the score a given user will assign to given movie. The key idea is that each movie has attributes, which must be combined with each users’ preferences. A PMF approach was applied to the Netflix problem by Salakhutdinov and Mnih (2008).
## Chapter 8

### Plan

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Start</th>
<th>Finish</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>First Year Report</td>
<td>15/09/2008</td>
<td>31/10/2008</td>
<td>7w</td>
</tr>
<tr>
<td>2</td>
<td>Implement Advanced Baseline</td>
<td>03/11/2008</td>
<td>30/12/2008</td>
<td>8.4w</td>
</tr>
<tr>
<td>3</td>
<td>Latent Cause Model Level 1</td>
<td>01/01/2009</td>
<td>30/01/2009</td>
<td>4.4w</td>
</tr>
<tr>
<td>4</td>
<td>Latent Cause Model Level 2</td>
<td>02/02/2009</td>
<td>27/02/2009</td>
<td>4w</td>
</tr>
<tr>
<td>5</td>
<td>Non-parametric Latent Cause Model</td>
<td>02/03/2009</td>
<td>01/05/2009</td>
<td>9w</td>
</tr>
<tr>
<td>6</td>
<td>Prepare NIPS 2009 Paper</td>
<td>04/05/2009</td>
<td>22/05/2009</td>
<td>3w</td>
</tr>
<tr>
<td>7</td>
<td>Binary PMF</td>
<td>25/05/2009</td>
<td>12/06/2009</td>
<td>3w</td>
</tr>
<tr>
<td>8</td>
<td>PC-Kalman</td>
<td>01/06/2009</td>
<td>01/07/2009</td>
<td>4.6w</td>
</tr>
<tr>
<td>9</td>
<td>DataPath Specific Issues for models</td>
<td>25/06/2009</td>
<td>01/09/2009</td>
<td>9.8w</td>
</tr>
<tr>
<td>10</td>
<td>Implement Some Application Code</td>
<td>15/07/2009</td>
<td>15/09/2009</td>
<td>9w</td>
</tr>
<tr>
<td>11</td>
<td>Write Paper for AI Stats</td>
<td>15/09/2009</td>
<td>15/10/2009</td>
<td>4.6w</td>
</tr>
<tr>
<td>12</td>
<td>Extend Models for Multi-Task Learning</td>
<td>15/10/2009</td>
<td>26/02/2010</td>
<td>19.4w</td>
</tr>
<tr>
<td>13</td>
<td>Write ICML 2010 Paper</td>
<td>15/12/2009</td>
<td>15/01/2010</td>
<td>4.8w</td>
</tr>
<tr>
<td>14</td>
<td>Implement Application Code</td>
<td>01/01/2010</td>
<td>01/04/2010</td>
<td>13w</td>
</tr>
<tr>
<td>15</td>
<td>Write NIPS 2010 Paper</td>
<td>15/04/2010</td>
<td>31/05/2010</td>
<td>6.6w</td>
</tr>
<tr>
<td>16</td>
<td>Write Thesis</td>
<td>03/05/2010</td>
<td>30/12/2010</td>
<td>34.8w</td>
</tr>
</tbody>
</table>

Figure 8.1: Gantt chart for PhD research. Level 1 of the Latent Cause Model refers to the autoregressive version while level 2 refers to hidden causes.

The goal of my research is to create models for multivariate time series. In some cases, it will even be multi-task as there could be several sets of multivariate time series. Particular attention will be paid to whether the models are compatible with data that contains rare events. A detailed and prioritized Gantt chart for my research plans is
shown in Figure 8.1.

The baseline metrics presented in Section 3.4 are good to get an idea of what performance a naive approach attains. However, before getting too sophisticated with new models it would be good to exhaust more options with regard to better feature extraction techniques and existing algorithms. This task, advanced baseline, is one of the first tasks in the Gantt chart. This task could also be of great practical use for DataPath. Some of the techniques used in the advanced baseline, such as EPCA, will be recent developments, but not novel approaches.

The latent cause model is an interesting model and could serve as topic for a NIPS paper. It is designed to be of use for the DataPath problem. After establishing a baseline its performance could be rigorously evaluated. It is prudent to break tasks in to sub-tasks with milestones to make them more approachable. For this reason, the latent cause model has been divided into three different levels or complexity. Also, the higher levels of complexity will likely be much more computationally intensive. It would be beneficial to first establish a level of performance with the simpler models before developing more complex models. This way we can rest assured those extra CPU cycles are not being wasted.

The binary PMF should not take very long to implement given my experience with collaborative filtering while working at Google. However, that version had limitations on the dimensionality of the hidden variables, but those could be easily corrected. It will likely not be as useful for the DataPath data set as the latent cause model, but it should take less time. I applied matrix factorization to a time series at Google. So, it may be applicable to the DataPath data set. The Google data set had many more periodic trends, which is where the BPMF is probably most useful. In addition, the BPMF could serve as a topic for an AI Stats paper.

The PC-Kalman should be relatively simple. The simplest version will likely be learning in a LDS with a rank restricted output noise covariance matrix.

There will likely be idiosyncrasies of the DataPath data set that will need to be addressed when using these models. It would be good to set some time aside for tailoring these models for the DataPath problem. I may also need some time to implement any very useful algorithms in application code as opposed to MATLAB, which is better for prototyping. I may need time to learn how to interface to the MaxView API.

For any of the models which prove to be useful I would like to extend them for use in a multi-task learning framework. As mentioned in Section 3.3 there is a multi-task aspect to the DataPath problem. This and other algorithm extensions could be the topic for an
ICML 2010 or NIPS 2010 paper. After making multi-task extensions I could upgrade the application code to utilize this ability.

Finally, I have allocated several months to write my thesis.
Chapter 9

Conclusion

The DataPath problem has three distinct elements to it: large multivariate time series, rare events, and multi-task learning. The approach to solving the problem is practice oriented and focuses on the generative model approach. The time series element is important because the notion of time will be important in the specification of the generative model. The element of rare events is another reason why using a probabilistic modeling approach will be beneficial to avoid the problems of standard discriminant analysis and overfitting. The multi-task learning element is an area where hierarchical modeling could yield benefits.

Basic approaches have been explored to get a baseline level of performance such as using standard dimensionality reduction techniques and standard regression techniques. The performance suggests that there is some degree to which system faults can be predicted; however, the precision using basic methods is poor. Most of the warnings would be false. In due course, my research will explore how much better one can do using more sophisticated methods.

The extension to classic time series methods: ARMA, LDS, and HMMs, to more complex models SKF, FHMM, iHMM will be followed. Many of my proposed research ideas include the various levels of complexity seen in these extensions. Going from the autoregressive to latent variable version of the latent cause model is analogous to the extension of a Markov chain to an HMM. And the extension from the second to third, non-parametric level, is similar to going from a HMM to an iHMM. Finally, the element of multi-task learning will add another possible extension to my models. The element is of multi-task learning is reflected in the data set because we have several fault logs from different units that are configured in different ways. Statistical strength can be gained by using information from all the units to estimate the parameters while simultaneously
recognizing that the parameters could be slightly different in different units, or tasks. These ideas, as outlined in my plan, could be useful to implementing code in a Data-Path application. These methods could be used publishing papers in machine learning conferences and journals.

The latent cause model, and its various levels of complexity, seem to be the most promising for the DataPath data set. To fully explore all the extensions in the latent cause model, such as the non-parametric version, could result in some very deep research topics. I would consider the binary LDS to be the second most promising method. It doesn’t explicitly model the presumed generative process in a failure network as well as the latent cause model. However, this does not rule out its usefulness. The other methods, BMPF and PC-Kalman, could be useful as well.
Appendix A

More Data

To get a sense of the phenomenon over all the alarms in the data set I created a box plot for durations, Figure A.1, and inter-arrival times, Figure A.2. Each alarm has its own box plot so one can analyze the differences in behavior.
Figure A.1: Box plot for duration of alarms over all alarms. Time is measured in log10 scale
Figure A.2: Box plot for inter-arrival times of alarms over all alarms. Note that time is measured in log10 seconds.
Appendix B

Orbit Fun

If one extends the non-linear dynamical system in the Kalman orbit experiment further to include a feedback element to the external force one gets interesting results. The direction of the external force is still rotating uniformly like in the example before, but the force in the \( x \) and \( y \) direction is proportional to the velocity in the \( x \) and \( y \) directions, respectively. Charts are shown which plot the trajectory for different proportionality constants, \( \alpha \). It is interesting to note that when the feedback aspect to the external force is applied a clear pattern emerges; the data no longer swings back and forth in a seemingly random fashion.
Table B.1: Trajectory of a body in Kalman orbit experiment when the feedback controller is added. The force constants are $\alpha = 1, 3, 6, 15, 30, 60$. 
References


