Toward Automating and Systematizing the Use of Domain Knowledge in Feature Selection

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

William Christopher Groves

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Maria Gini

August, 2015

Acknowledgements

First and foremost, I would like to thank my thesis adviser, Maria Gini, for her timeless wisdom, boundless enthusiasm, and continuous support through my doctoral studies. She allowed me to have considerable latitude to explore broadly and to forge a research path that was truly my own. She is an excellent role model as a pragmatic researcher, mentor, and teacher. She is always willing and happy to provide advice, criticism, or encouragement in the right quantities and at the right moment to improve the overall outcome. In every research collaboration, she emphasized the importance of always understanding where the work fits in relation to the most simple benchmark, the optimal approach, and other approaches from the literature. Through her mentorship, I came to understand that the most important skill in research is in being able to accurately, simply, and clearly explain our design choices and research innovations to others. Innovation without a clear explanation is not useful and is often not an innovation at all. Her calm, wise, and methodical approach eases the uncertainties that naturally exist when tackling new research problems. Her approach is something that I will carry with me in all future endeavors.

I wish to thank my thesis committee: John Carlis, Maria Gini, Mohamed Mokbel, and Carlos Tolmasky. I very much appreciate my committee's valuable input and patience. Their questions and suggestions have helped greatly in carrying this work to its conclusion.

I would like to extend a sincere thank you to several students and researchers over the years who served as role models, provided inspiration for my efforts, or led me to a kernel of an idea that dramatically improved this work. This group includes Jon Rogness, Ivaylo Ilinkin, Sungkee Kim, Shana Watters, Steve Jensen, Wei Hsu, Wolf Ketter, John Collins, Luis Ortiz, and Michael (Tianhui) Li.

Dozens of students at the university have helped me to grow as a scholar, teacher and researcher. I would especially like to acknowledge the contributions of my fellow students in the Artificial Intelligence Research Lab including Mohamed Elidrisi, Steve Damer, Elizabeth Jensen, Ernesto Nunes, Julio Godoy, Marie Manner, Nick Johnson, Mark Valovage, and Hakim Mitiche. Your honest and sincere feedback on my research as it progressed have been very valuable. I will fondly remember our intense discussions in B22 that served both to entertain and to (hopefully) inform.

Thank you to my parents, their unwavering support and encouragement over the years have made this thesis possible. Finally, thank you to Xiaohua Yang for endeavoring to understand my unfinished ideas far from your own field of study, being my coach, and being continuously encouraging of my studies.

Dedication

To my parents

Abstract

Constructing prediction models for real-world domains often involves practical complexities that must be addressed to achieve good prediction results. Often, there are too many sources of data (features). Limiting the set of features in the prediction model is essential for good performance, but prediction accuracy may be degraded by the inadvertent removal of relevant features. The problem is even more acute in situations where the number of training instances is limited, as limited sample size and domain complexity are often attributes of real-world problems. This thesis explores the practical challenges of building regression models in large multivariate time-series domains with known relationships between variables. Further, we explore the conventional wisdom related to preparing datasets for model calibration in machine learning, and discuss best practices for learning time-varying concepts from data.

The core contribution of this work is a novel wrapper-based feature selection framework called Developer-Guided Feature Selection (DGFS). It systematically incorporates domain knowledge for domains characterized by a large number of observable features. The observable features may be related to each other by logical, temporal, or spatial relationships, some of which are known to the model developer *a priori*. The approach relies on limited domain-specific knowledge but can replace or improve upon more elaborate domain specific models and on fully automated feature selection for many applications. As a wrapper-based approach, DGFS can augment existing multivariate techniques used in high-dimensional domains to produce improved modeling results particularly in situations where the volume of training data is limited. We demonstrate the viability of our method in several complex domains (natural and synthetic) that have significant temporal aspects and many observable features.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The task of learning concepts automatically from data is changing rapidly in the era of "big data". More and more data is being collected about natural phenomena, social processes, and business decisions than ever before. It is essential to find new methods and application domains to fully unlock the new opportunities for improved decision-making enabled by this new era of knowledge. This work is a step toward developing systematic learning techniques for domains with many relevant sources of information. The contributions here can systematically augment already very powerful domain independent learning algorithms with additional domain knowledge. The existing body of work on machine learning models is extensive and many methods have desirable theoretical properties. We propose to leverage these existing benefits while simultaneously improving models built for difficult learning scenarios by leveraging domain knowledge. This allows for more accurate models in challenging scenarios in which there are both many potential inputs and when the amount of observation instances is limited (possibly due to availability challenges or expense).

Traditional machine learning models applied to real-world data involving many data sources ("multivariate domains") perform best when all variables used in prediction are relevant and informative. Early prediction models were driven significantly by human expert advice both in model selection (the approximating function) and in feature selection (the variables provided to the function). As the scope of available data and available processing power have both increased, both general algorithms and automated feature selection methods have come to the forefront.

Feature selection is often formulated as an optimization process that prunes the set of input variables prior to calibrating the final learning model used for prediction. Some machine learning algorithms perform feature selection internally by implementing a decision model that only considers a subset of the features (decision trees and "LASSO" are examples of such methods). Some feature selection methods are external to the learning model and do not consider

the learning model at all for determining a feature subset. Automated methods can give the impression that there is no need to limit the variables used for learning as excess features can be pruned by feature selection (external or internal). However, even with automated feature selection, including all potentially relevant variables in the feature set can degrade prediction performance when compared to using a more minimal and relevant subset. External feature selection methods from the literature are often fully automated (except for some tuning parameters) and consider all variables as equally likely to be predictive. In practice, all variables are not equally informative. There is significant information contained in the relationships between variables in many domains. For example, Bayesian networks are a method where a prior distribution is used to encode such knowledge.

To leverage this information in feature selection, we propose a framework for feature selection designed for domains that have a strong temporal or spatial component with a large number of (possibly) related variables. The framework, called *Developer-Guided Feature Selection* (DGFS)[1], systematically incorporates domain knowledge into the feature selection process to improve performance of the prediction model. This method is valuable in cases where learning efficiency is a concern, such as when the number of training samples is limited or the number of feature set evaluations must be constrained. A diagram showing the evaluation process is shown in Figure 1.1.

We validate this approach with empirical results in four domains:

- First, we present experiments in an artificial data domain for the purpose of testing the properties of our method. Synthetic data provides great opportunities to systematically study the power of the proposed methods in a controlled environment. We encode specific relationships in a noisy multivariate dataset and embark on a model recovery experiment to show the efficacy of our approach. Variable relevance and time-offset relationships can also be inferred based on the results of the feature selection search. The discussion outlines the additional domain knowledge to be discovered through the feature selection search process.

  This domain mimics the spatial and temporal properties of real-world time series data. In particular, two artificial datasets are used to examine (1) the effect of reduced training set size (which can make calibration harder) and (2) the effect of many irrelevant variables in prediction. As these are artificial datasets, long observation sequences are generated, and many complete datasets generated from unique initial random seeds are used in the analysis. The ability to generate samples with no limits allows for thorough statistical analyses to be performed when comparing methods (which is often not possible in natural

[1] In our previous work [Groves and Gini, 2013b], this framework is called User-Guided Feature Selection. The change more accurately reflects the source of knowledge used in the bias.

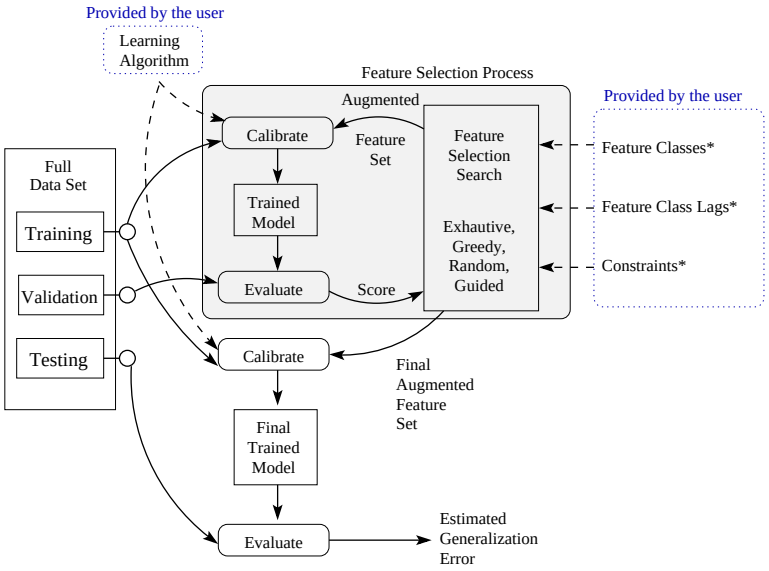


Figure 1.1: A diagram showing the information flow in the feature selection search process using Developer-Guided Feature Selection (DGFS). Dotted lines show domain knowledge provided by the practitioner when building a model. A "$*$" denotes information specific to model construction with DGFS.

domains).

- The second application shows the promise of the proposed method in a decision making context. In the real-world domain of airline ticket price prediction, we show how the method can reduce consumer purchase costs. While there are existing studies of the airline ticket purchasing task, there are many limitations of the existing works. Our methods significantly extend the state-of-the-art in this domain in terms of prediction performance (with the objective of reducing costs for customers) and in terms of the specificity of the models (by showing the effect of modeling specific user preferences such as non-stop only flights on a specific airline). These prediction improvements are achieved in our method by grouping the many observable variables in the domain based on specificity (i.e. how much aggregation is involved in each statistic). The feature selection process balances complexity with accuracy in a domain with few observation instances (≈ 250) in relation to the number of variables (≈ 92). The computed models achieve an average purchase cost reduction of 7% over the naïve ("earliest purchase") strategy.

  This natural data domain involving optimal airline ticket purchasing from the consumer's perspective is challenging principally because buyers have insufficient information for reasoning about future price movements. Our model uses historical price quotes to compute

expected future price statistics for all available flights on specific routes and dates. Additionally, we extend the modeling to predict itineraries with specific desirable properties such as flights from a specific airline, non-stop only flights, or multi-segment flights. These models are facilitated by a large corpus of data collected from an automated data collection process that yielded a daily price quotes for a 109 day period from all airlines for 7 different origin-destination pairs (including both US domestic and international routes). Each query returned on average 1,200 unique round trip itineraries from all airlines; most queries returned results from more than 10 airlines. Because of the high level of detail of this data set, it is possible to leverage price trends as well as competitive pricing relationships between airlines to improve prediction. Our results show that users can lower the average cost of purchases in the 2 month period prior to a desired departure. Our method compares favorably with a deployed commercial web site providing similar purchase policy recommendations [Groves and Gini, 2015].

- The third domain is from the real-world application of river/stream flow prediction. Human activities can effect the natural environment and efficient modeling of these changing natural processes is an important contribution for improving decision making. DGFS is utilized to find a regression model that incorporates multiple data sources (stream flow measurements and future precipitation predictions) and time lagged variables. The model has improved prediction accuracy over conventional machine learning models. We make predictions about flows on the Mississippi River network using hourly flow measurements and precipitation data from the US NOAA (National Oceanic and Aeronautical Administration) using two years of data.

  This domain is characterized by a strong interaction of spatial and temporal features. In this domain, predictions are made for future river flows from 12 to 120 hours ahead at a specific measurement site in the river network. Previously observed flow information at upstream sites as well as predicted precipitation information are the features available for prediction. Physical relationships between sites are the source of the constraints leveraged to improve prediction. The dataset contains over two years of flow and precipitation prediction samples at one hour resolution at each of the measurement sites.

- Our fourth application is on price prediction in a multi-agent supply chain management simulation and competition. Decision making for manufacturing is becoming more systematic (e.g. by employing statistical techniques for determining optimal decision making). Future needs in sales, production, and procurement can be anticipated through automated analysis of the changing market environment. The Trading Agent Competition in Supply Chain Management is a multi-agent competition which is designed to test fully automated

business strategies. Our experiments in this domain involve estimating future prices for products based on price information from 10 manufacturing inputs and 15 manufactured product outputs. All of these prices are themselves varying over time due to market forces that are emergent from the actions of other manufacturers and from environmental stochasticity. The DGFS framework is applied to find a relevant feature subset that improves price prediction accuracy. This approach finds statistically significant improvements in future (20 days ahead) manufacturing input prices that are better than existing published state-of-the-art approaches for this domain. This domain is well studied by others in the development of automated trading agents, but this prediction problem is challenging due to the large number of possibly relevant variables, few observations (220 time units per simulation which approximates one year of daily decision making), and rapid price changes due to competition.

Economic analysis [Groves et al., 2009] and prediction [Groves and Gini, 2013a] relevant to the Trading Agent Competition for Supply Chain Management (TAC SCM) is a challenging application domain due to the many specially tuned processes already implemented for competition. TAC SCM is a multi-agent supply chain simulation involving an oligopoly of competing, fully autonomous agents who seek to maximize profit. In the simulation, agents purchase component parts from suppliers, construct computers with the component parts, and sell the computers to customers. This is a complex simulation competition with an annual tournament that has attracted since 2003 teams from around the world [Collins et al., 2006]. In each tournament year, there are many (usually 18) repeated tournament rounds with the same mixture of manufacturing agents but with different stochastic environmental conditions. Data available from a single agent's perspective in these repeated encounters is used as the data source for these experiments. A stylized information flow diagram for the supply chain domain is shown in Figure 1.2.

The key features of our Developer-Guided Feature Selection (DGFS) method are (1) that instances of time-delayed features are included, (2) a domain-specific feature similarity hierarchy is leveraged, and (3) the features used in prediction are added or pruned based on in-situ performance. Because of these novel features, our method is able to perform well relative to other machine-learning models on high-dimensional datasets when the amount of training data is limited.

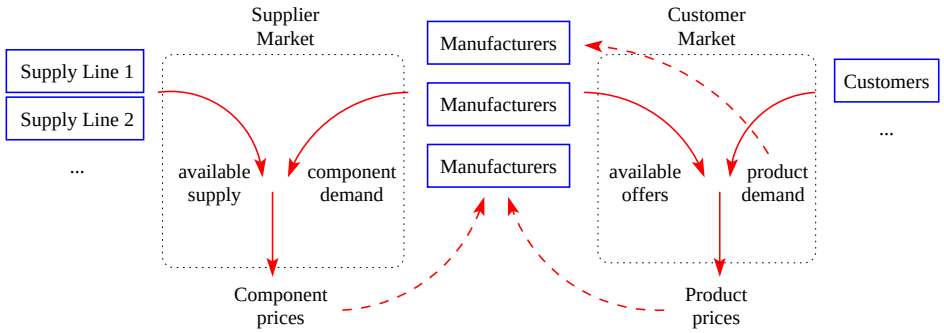


Figure 1.2: A diagram showing the information flow in the supply chain domain. The solid lines represent complete information flows, and the dotted lines represent indirect or partially observable information flows.

1.1 Contributions

The essential contributions of this thesis include:

- Providing an overview of feature selection methods applicable to regression and classification on real data. While our focus is on the regression case, many methods applied to classification are also relevant to feature selection in regression, so they are also included.
- Measuring the effectiveness of using time-delayed observations as elements in the feature vector for learning.
- Demonstrating feature categorization using a feature class hierarchy which guides the inclusion of variables into the feature set selection process.
- Extracting domain knowledge from the resulting models (both from the model parameters, depending on support from the model, and from the selected feature set).
- Discussing fully-automated approaches to feature selection that have become popular in the literature for use in domains with many variables.
- Systematically demonstrating the use of a hierarchical feature set on temporal data.
- Showing how hierarchical feature set selection can contribute to improved prediction modeling (over fully automated, domain-independent methods) in airline ticket prices, river/stream flow, and supply chain domains. These domains are characterized by 1) known relationships (possibly spatial or logical) between variables, 2) a large number of variables potentially relevant to prediction, and 3) a relatively small number of observations relative to the number of variables.

- Discovering, specifically in the airline ticket domain, general airline pricing principles based on specific airlines and purchase preferences.

1.2 Chapter Summary

- Chapter 1 introduces the analytic goals pursued in this thesis. We motivate our discussion and analysis approach.
- Chapter 2 discusses related work both for general feature selection approaches and for prediction models in the domains studied.
- Chapter 3 provides the algorithmic details of the proposed feature selection framework used in the application domains.
- Chapter 4 introduces a synthetic data domain which is used to experimentally validate properties of the proposed method. We examine topics including sensitivity to non-stationary concepts, training set size, and irrelevant variables. This domain mimics the relationships observed in both the physical and the economic real-world application domains.
- Chapter 5 applies the proposed DGFS framework to airline ticket price prediction.
- Chapter 6 introduces the river-flow application domain. This domain has significant spatial and temporal relationships between features.
- Chapter 7 demonstrates DGFS applied to price prediction in a supply chain domain.
- Chapter 8 concludes the thesis with thoughts on the future of feature selection with developer-guided and data-driven methods.
- Appendix A provides extended experimental results for the synthetic data domain.
- Appendix B shows additional result details from the river flow domain for the interested reader.

Chapter 2

Literature Review

This chapter provides a general survey of feature selection approaches relevant for classification and regression in real-world application domains. Much of the existing work on large multivariate problems is in the context of classification (i.e. assigning a class label to each sample), but many of these principles are also readily applicable to the regression context (i.e. assigning a real or integer value as a label to each sample) as well. On the topic of feature selection, we examine the available data and the types of variable relationships that are commonly observed in real data.

We later survey the learning models that have been applied to real-world multivariate domains. Feature selection can also be posed as an algorithm parameter setting task ("hyperparameter optimization"), and we discuss the state-of-the-art approaches for this process. Finally, the design of the experimental protocols is also discussed related to both the dataset splitting (cross-validation) approach and prediction model evaluation.

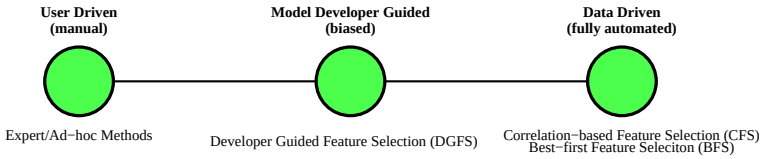
2.1 Feature Selection

The process of feature selection involves finding a subset of features (variables) from the original set that is then used for machine learning from data. The intent is to find a feature subset that is more desirable than the original large set. For a comprehensive overview of the historical context and of the range of existing approaches see [Guyon and Elisseeff, 2003] and [Hall, 1999].

Methods of feature selection in machine learning exist along a continuum based on the degree of automation and developer input required (Figure 2.1). Studying the degree of automation is a particular focus of this thesis.

In the simplest case, the developer encodes a feature selection implicitly in the choices of features provided to the learning algorithm (i.e. a developer driven approach). The developer

Figure 2.1: Feature selection continuum based on the degree of developer input



may tune the model by experimenting with additions and removals of features until some performance level is reached. The choice of learning model is often varied in the quest for improved performance as well. Domain practitioners often have knowledge of the basic underlying physical or social processes which generate the data, so they may be well suited to perform some optimization in both model choice and feature selection.

At the other end of the spectrum, feature selection can be handled by a specialized algorithm for this purpose. This is particularly useful in domains where there are many features (100s or 1000s) that are potentially relevant to the prediction. Some amount of feature selection may be performed implicitly in the chosen learning algorithm through a weighting of variables.

This work proposes a middle path in which domain knowledge about relationships between features is incorporated in a systematic way with an explicit feature selection process, called Developer-Guided Feature Selection (DGFS). But first, we will discuss advances in fully automated methods that consider only the dataset itself.

2.1.1 Data Driven Feature Selection

In the machine learning literature, considerable effort has been directed towards building robust models and calibration methods that work well in large multivariate domains and are resistant to the presence of non-predictive variables. These learning algorithms perform feature selection internally by building models that operate over only a subset of the variables. In the terminology of Hall [1999], these are called embedded methods. Additionally, there are two types of explicit methods performing feature selection as a separate process: filter approaches (the feature set chosen is independent of the learning algorithm used), and wrapper approaches (the feature set is chosen through performance evaluation of the calibrated learning algorithm). Table 2.1 briefly compares these three categories of data driven feature selection methods.

2.1.2 Black-Box/Wrapper Methods

Wrapper-based feature selection refers to methods that use the final learning model as a black-box to measure the fitness of candidate feature subsets. This is implemented often as an outer-loop optimization process involving many evaluations of candidate subsets to arrive at a good

Table 2.1: Taxonomy of data driven feature selection methods.

Wrapper Methods	• Measures performance of feature set using chosen ML algorithm and picks the set using the measured performance. • Survey Reference: [Kohavi and John, 1997] • Algorithms: Best-First Search [Kohavi and John, 1997], Simulated Annealing, Genetic Algorithms, Random Forest, Developer-Guided Feature Selection [Groves, 2013]
Filter Methods	• Feature selection is independent of ML algorithm. • Survey Reference: [Hall, 1999] • Algorithms: Variable Ranking, Correlation-based Feature Selection [Hall, 2000], RELIEF [Kira and Rendell, 1992], one (simple) algorithm as filter for another (complex)
Embedded Methods	• Feature selection is performed implicitly within a learning algorithm. • Survey References: [Guyon and Elisseeff, 2003; Hastie et al., 2001] • Algorithms: Decision Trees, Neural Networks, LASSO, ...

candidate. At a high level, these methods generate a candidate feature subset which is then used to train a model. The model is then evaluated. This evaluation is repeated many times until some stopping criteria is met. The best observed candidate subset is then chosen as the final augmented feature set. The benefit of this approach is that the evaluation criteria is relevant to the learning algorithm. This is in contrast to the filter-based approach: the learning algorithm (e.g. multivariate regression) may be very different from the feature selection criteria (e.g. correlation).

The Developer-Guided Feature Selection (DGFS) method, presented in this thesis, is a wrapper-based approach. In the feature selection process, many candidate feature sets are evaluated using the chosen learning algorithm. These evaluations are used to determine the final feature set. From the literature, the Best-first Feature Selection (BFS) algorithm is also an example of a wrapper method and uses a similar evaluation process [Kohavi and John, 1997]. DGFS extends BFS in several ways: first, by constraining the search space of candidate feature sets, second, through the choice of search process, and third, through the addition of time shifted variables to the feature set. Both of these methods are examined in detail in Chapter 3.

It is important to note that feature selection can also be formulated as an algorithm parameter (hyperparameter) optimization process as well (see also Section 2.4). There are many parallels between these classes of methods.

2.1.3 Filter Methods

Filter-based feature selection uses an inexpensive to compute heuristic that is learning algorithm independent to construct a feature subset which is then passed to the learning algorithm. Filter methods are machine learning algorithm agnostic and are not tuned for any particular algorithm.

Also, there is no need for a validation set (unlike in the wrapper approach), so the validation set instances can be used to supplement the training set instances.

Filter methods often correlation-based and, therefore, work best on categorical or binary variables. The RELIEF algorithm (in the binary classification case) measures the relevance of each feature relative to the target using correlation Kira and Rendell [1992]. It is a randomized algorithm that iteratively selects a training sample. It updates the relevance weight of each feature using distance information computed from the nearest same-class sample and the nearest different-class sample. Features are relevant if they exceed a relevance threshold. This method is simple, but it cannot handle dependencies between features. For example, multi-feature dependencies present in a two-variable XOR problem cannot be handled with this approach Guyon and Elisseeff [2003].

In Correlation-Based Feature Selection (CFS), the heuristic used is Pearson's correlation which is measured both between independent variable-dependent variable pairs and between pairs of independent variables. It measures the fitness of subsets of variables in its search process [Hall, 1999]. The final feature set used for learning is found through a best first search of possible subsets. The final feature set does not depend on the choice of machine learning algorithm which may be an advantage because the final algorithm used can be very expensive to calibrate.

This approach is not optimal in general and cannot determine dependencies between variables: it assumes the covariance matrix between independent variables is diagonal. One advantage is that this approach is readily parallelizable is inexpensive to evaluate (a worse case of $O(n^2)$ in terms of the number of variables but each evaluation is inexpensive and need only be performed once during the search). This method is used as a benchmark and is discussed in implementation-level detail in Chapter 3.

2.1.4 Embedded Methods

Learning algorithms with embedded feature selection are methods that build a model that only uses a subset of the input variables to compute the response. Many popular machine learning algorithms have this property including decision trees, artificial neural networks, and LASSO.

The embedded approach to feature selection has several advantages. Embedded methods can use the full training set for calibration and avoids the use of an out-of-sample validation set. By calibrating using as much data as possible for training, the model may be able to learn concepts more fully than wrapper methods which require a hold out set to evaluate candidate subsets. Also because of the tight integration between feature selection and learning, the learner may be more computationally efficient than filter or wrapper methods.

Feature selection and learning model complexity are related. In machine learning, the expressive power of the model is well understood for many algorithms. Even from simple components, the expressiveness of many models can fully encode an almost unlimited range of relationships between independent variables and the dependent variable. For example, neural network models of sufficient complexity, in terms of the number of hidden nodes and layers, can encode any arbitrary mathematical function [Russell and Norvig, 1995]. The central challenges of applied machine learning are in 1) choosing the model to calibrate, and 2) to calibrate the model successfully given (possibly limited or noisy) data.

In the general area of Statistical Learning Theory, the concept of Structural risk minimization (SRM) addresses the tradeoff between goodness-of-fit of the model to the training data and the complexity of the model [Vapnik, 1998]. SRM prescribes that models should be evaluated using a single loss measure that encompasses both concepts:

$$\text{model score} = \text{empirical risk} + \text{complexity penalty} \tag{2.1}$$

Applying the statistical learning theory concepts also allows for comparisons of models with different structures or construction techniques. Two general measures of machine learning algorithm complexity are degrees-of-freedom and VC (Vapnik-Chervonenkis) dimension. Degrees of freedom (*df*) for a linear regression model is computed as the number of parameters of the model m plus one:

$$Y = \beta_1 X_1 + \beta_2 X_2 + \ldots \beta_m X_m, \tag{2.2}$$

where β_i is the weight coefficient for variable X_i. VC dimension is an alternative measure that can more accurately measure the expressive power of a model [Abu-Mostafa, 1989]. It is a measure of the maximum number of dichotomies (mutually exclusive divisions of d points) that the model is able to divide in the classification case. A model with d degrees of freedom is able to divide $d + 1$ specifically placed points in the hyperspace and arbitrarily assign specific labels (e.g. +1 or -1) to each of them. The VC dimension of many popular methods is known: the value can be computed exactly based on the model structure for linear regression, decision tree, and SVM; VC dimension is infinite for k-nearest neighbor; no accurate estimate is known for non-linear neural network models.

A common approach for balancing model complexity and model parameter tuning is to fix one and optimize the other. For example, if the degrees of freedom is set a priori in the calibration process, the calibration attempts to find the best model given this limit on the model complexity [Guyon and Elisseeff, 2003]. As all models have the same limit on complexity, making model comparisons based on accuracy is valid. Choosing the best among these based on accuracy is a reasonable choice. SRM is relevant to the methods proposed in this thesis because

1) we similarly fix or limit number of features as input to the ML algorithm and 2) we employ ML methods with a fixed number of degrees-of-freedom. These aspects allow direct comparisons among models of different composition.

Decision tree models are an embedded method. The prediction label for each observation to predict is often computed using just a small subset of the available variables. Decision trees are most commonly applied in the classification case but can also be used for regression. The straight forward approach of regression using decision trees (e.g. using the C4.5 algorithm) involves computing the response value for each leaf node as the mean of the samples belonging to the node [Loh, 2011]. Decision trees have many advantages: the method is robust to irrelevant variables (irrelevant variable are not included in decision nodes), the resulting model can be easily understood, it can natively handle missing values in the input, and it can model conditional dependence between variables. Trees can be sensitive to the input data (small changes to the input can result in large changes to the output) and the performance can be degraded when there are few training samples relative to the number of variables. Alternative methods that have a smoother response model tend to perform better in the regression case. Decision trees are well suited for modeling non-linear responses because the model can react non-linearly to small changes in the input variables.

Ordinary Least Squares (OLS) is a linear regression model in which the target variable is computed as the sum of weighted input variables. It finds the weights by minimizing the sum of squares of errors for each training set observation (Equation 2.3). Least squares is a standard solution for overdetermined systems (more equations than unknowns). OLS provides a simple, explainable model. However, it cannot automatically fit data for which the target variable is not the result of a linear combination of input features, and it is highly sensitive to errors in the input dataset and prone to overfitting.

$$\min_x f(x) = \underbrace{\|A\beta - b\|_2^2}_{\text{residual error}}, \tag{2.3}$$

where A is the training matrix, b is the vector of actual true labels, β is the weight vector for the regression model, where $\|\bullet\|_2$ is the Euclidean norm.

Regularization is the process of adding a penalty term to the minimization which incorporates additional information about the problem to prevent overfitting, to enable solving of an ill-posed (i.e. singular) problem, or to bias the parameter optimization using a distribution prior. Ridge regression is one type (using the L_2-norm over the variable coefficients). Ridge Regression (Equation 2.4) addresses some problems of OLS by imposing a penalty (or regularization term). Ridge regression uses a L_2-norm regularization which biases the coefficient weights to be close to zero. The regularization parameter α can be optimized for a given prediction domain through

cross-validation.

$$\min_x f(x) = \underbrace{\|A\beta - b\|_2^2}_{\text{residual error}} + \underbrace{\alpha\|I * \beta\|_2^2}_{\text{regularizer}}, \tag{2.4}$$

where I is the identity matrix. Ridge can reduce overfitting and make underconstrained systems solvable. But the optimization produces all non-zero coefficients (i.e. no feature selection occurs) and is effected by outliers.

LASSO (Least Absolute Shrinkage and Selection Operator) is another linear regression algorithm which instead adds an L_1-norm penalty over the variable coefficients (Equation 2.5). More generally, this can be used to add a prior distribution over the choice of parameter values and specifically biases the coefficients to be exactly zero [Tibshirani, 1996].

$$\min_x f(x) = \|A\beta - b\|_2^2 + \alpha\|\beta\|_1, \tag{2.5}$$

where $\|\bullet\|_1$ is the Manhattan norm. LASSO incorporates feature selection implicitly due to the assignment of many zero-value coefficients and it is not strongly effected by outliers because the optimization process converges to median. However, LASSO is more expensive to compute than OLS or Ridge because it cannot be solved in closed-form due to being non-convex.

Hastie et al. [2001] present a framework for decomposing the sources of prediction error when calibrating a model from data. A trained model will invariably be some distance from the true model when learning from data due to noise and also due to bias induced both from the model choice as well as from the model estimation process (Figure 2.2). In the case where the model can be of arbitrary complexity (any number of features, internal parameters, etc.), the variance of the estimation process can be large. By restricting the model space further (e.g. by reducing the number of features), it is possible to reduce this variance. This restriction is worthwhile (the mean error is reduced when predicting on new samples) if the decrease in variance exceeds the increase in the square of the bias. This is applicable to machine learning in general either with or without feature selection as an explicit step.

Many methods also adjust the relative importance of the input features which is effectively a kind of embedded feature selection. Artificial neural networks are an example of this. The methods assign weights to each input in the model network which sets the contribution of each variable.

Economic data generally features a large number of simultaneously observable variables that contain potentially relevant information for prediction. Many types of multivariate algorithms can capture relationships between sets of variables that can be used for prediction. [Martens and Næs, 1992] provides an excellent overview of multivariate techniques. There are also many types of time series prediction methods that perform prediction for temporal data.

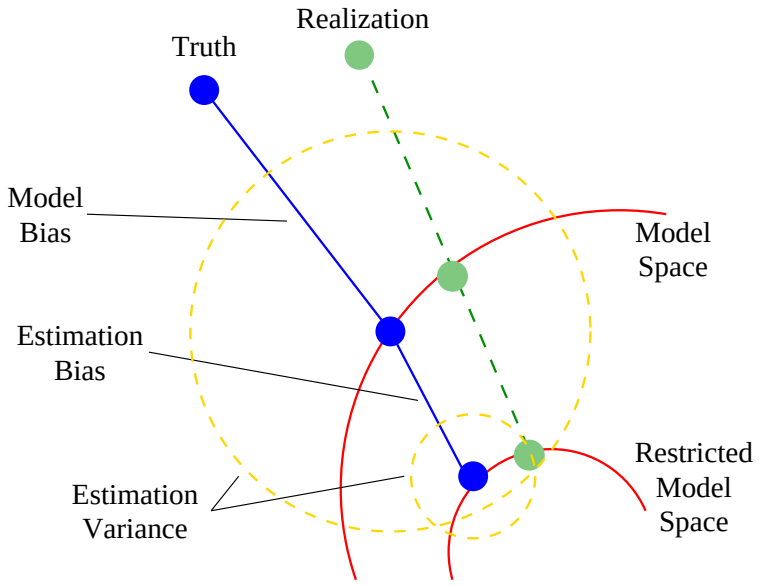


Figure 2.2: A stylized diagram showing sources of error in model estimation from data. By limiting calibration to a subset of the model space that also has reduced model complexity, it is possible to reduce the variance of the constructed model [Hastie et al., 2001].

2.2 Application Domains for Feature Selection

We present an overview of prediction techniques applied to several domains where feature selection would be beneficial for dimensionality reduction. We seek both to examine the challenges in real-world prediction domains and to examine the prediction approaches that perform well. As time series prediction is a topic with a long history, the focus of our analysis is on domains similar to the airline ticket price and supply chain domains.

2.2.1 Airline Ticket Prices

Airline ticket pricing is a complex but promising domain for price prediction because it possesses several distinct market characteristics. In the airline ticket price domain, Etzioni et al. [2003] compare the effectiveness of data mining (Ripper and Q-learning), time series, and expert knowledge rule-based prediction techniques on prices for a specific set of airline flight itineraries. The primary purpose of prediction in their work is to provide guidance on optimizing the timing of ticket purchases based on historical trends. While this domain clearly possesses a spatial component (e.g. the airline network can be represented as a graph or the physical distance of each itinerary are spatial in nature), the spatial aspects of the prediction task are not directly considered. This is likely because the non-spatial aspects of the prediction domain including

airline network operational structure, traffic density, and long-term strategic decision-making are more significant factors in price behavior. The variables for each daily price observation that are condition the model output are the current price, number of hours until departure, and identifying information about the flight (route, flight number, and airline).

Bachis and Piga [2007] address the spatial aspects of the airline ticket price domain in search of price arbitrage relationships. The authors find cases of persistent mispricing by airlines serving continental Europe. In these cases, customers can reduce their costs by buying individual flight legs of a trip instead of purchasing an entire multi-leg trip as a package. In particular, the authors provide evidence that these mispricing events are prevalent due to price discrimination based on the currency and geographic origin of the price quote. In related work, it is found that there exists an artificial price premium for itineraries starting or ending at "hub" airports. This indicates that market power plays a significant part in airline pricing strategy [Bachis and Piga, 2006].

2.2.2 Supply Chain Management

Because studying detailed information from a real supply chain is difficult due to its specialized and proprietary nature, there has been a significant body of research on a highly-detailed simulated multi-agent supply chain scenario that mimics the complexities of real-world supply chain management [Sadeh et al., 2003]. The advantage of studying a simulated scenario lies in the ability to fully instrument the underlying processes as they happen for later analysis and to be able to rapidly test novel management strategies.

The current iteration of the Trading Agent Competition for Supply Chain Management (TAC SCM)[1] simulates a one-year product life-cycle (modeled as 220 days) in a three-tier supply chain, including parts suppliers, end customers, and a set of competing manufacturing agents. Each agent must purchase parts in a competitive procurement market, manufacture finished goods, and sell them in a competitive sales market. This kind of market scenario is common to many fast moving electronic goods makers, such as the Dell Computer Corporation. Significant effort has gone into producing top-performing agents in the competition (for instance, [Benisch et al., 2004, 2009; Kiekintveld et al., 2006; Pardoe and Stone, 2007; Stan et al., 2006]).

In the supply chain scenario, there are two different markets where price prediction is relevant. In the component market, agents are purchasing parts from suppliers. And in the customer market, agents are selling finished goods to customers. The two markets differ in three fundamental ways: 1) agents are selling in the customer market but are buying in the component market, 2) the lead time[2] of requests made in the component market is specified by the agents,

[1] For more information on TAC SCM, please see http://tac.cs.umn.edu/ .

[2] Lead time is the amount of time (e.g. in days) between when a request is made and when the goods are

and 3) the range of lead times can be much longer in the component market (1 to 200 days) than in the customer market (3 to 13 days). Several agents use highly tuned models for predicting future customer market prices because this information is needed to drive a profitable sales strategy. The methods used to predict supply market prices tend to be less sophisticated than the methods used for customer market predictions, but the competitors have addressed the supply market prediction using a variety of approaches.

For instance, the TacTex agent [Pardoe and Stone, 2007] relies on a domain specific approach to component price prediction by modeling the latent variables used in the actual supplier price computation and making requests that maximize the confidence in the latent variable estimates. In domains where the underlying price generation process is not known, this kind of method has no traction. While the prediction method is not generalizable, the notion that, in situations where the ability to make observations is constrained or costly, requests should be specifically crafted to maximize observability is a useful approach.

Other agents in this application domain use non-parametric methods for prediction. Benisch et al. [2006] introduce a *nearest neighbor* (kNN) approach to computing supplier prices by taking set of recent price observations seen by the agent and computing price forecasts for possible future component requests. This prediction technique has been adopted by other top teams in the competition as well [Collins et al., 2009; Kiekintveld et al., 2007]. This is a model-free on-line prediction method that attempts to address the lack of sufficient observed data density over the range of lead-times.

The DeepMaize agent [Kiekintveld et al., 2007] uses an on-line learning method over recent observations to compute a linear interpolation over the lead-time range from the price observations seen. This line of best fit is then fed into a decision tree classifier, trained off-line on a large corpus of previous games, which produces an accurate price prediction function. It should also be noted that the decision tree classifier also stores sufficient data to make predictions about changes in prices many days in the future. This model enables the agent to predict how component prices will vary into the future, and therefore, provides sufficient information to decide to defer or accelerate procurement scheduling based on the predicted changes in prices. The prediction model used in DeepMaize is the most expressive prediction model of any published TAC SCM agent architecture.

In many of these agents, domain specific features of the market are used to guide or improve the predictions.

scheduled to be delivered.

2.2.3 TAC Travel

TAC Travel is a multi-agent travel brokering tournament in which a group of 8 competing agents must purchase airline tickets, hotel rooms, and entertainment tickets in response to customer requests. Each agent does its best to procure complete travel packages for its small set of customers that best align with stated customer preferences and at lowest cost. Each type of ticket has its own separate market and the agents must trade in all three markets to successfully fulfill customer requests before each tournament completes.

While the TAC Travel agent competition seems to be a reasonable point of comparison for price prediction methods in a multi-agent environment, several aspects of the competition make the methods employed not directly transferable to a temporal, dynamic pricing scenario. Analysis of this game has shown that there is little useful information revealed to agents during the tournament run so the game functions similarly to a one-shot tournament.

In particular, the hotel room bidding portion reduces to a sealed-bid auction occurring at the end of the game: each agents best strategy is to defer bidding their true preferences until the last possible moment [Stone and Greenwald, 2005]. The prices quoted in the airline ticket market are insensitive to competitive agent behavior and the entertainment market, while inextricably tied to agent behavior, does not significantly effect the outcome. This lack of significant additional information revealed in the middle of each game run makes the price prediction approaches used in this domain less interesting. the Many agents form price predictions of the final hotel prices based on the initial conditions of each game. These predictions are often based on historical information from previous games, adjusted using the current initial conditions, and form the basis of decision-making.

The analysis of Wellman et al. [2004] specifically considers the mechanisms agents use to predict final hotel auction prices in the TAC Travel tournament. The authors' analysis finds that tournament agents used three categories of techniques for the hotel prediction task: historical, machine learning, and competitive analysis. Each of the three types performs as follows: the historical type performs basic aggregation of prices in previous games, the machine learning type formulates price prediction as a machine learning problem using observable initial airline ticket prices as features, and the competitive analysis type uses an idealized model of the scenario to compute hotel prices at the competitive equilibrium.

This is in contrast to the TAC SCM scenario where recently observed information of the current market condition is significantly more important for decision-making than aggregate historical data from previous games. However, given the repeated nature of this game, it is a valuable point of comparison.

2.2.4 Stock Market Trading Simulation

Another real-time multi-agent competition whose competitors employ price prediction in some fashion is the Penn-Lehman Automated Trading (PLAT) competition [Kearns and Ortiz, 2003]. The PLAT was a market micro-structure simulation designed to examine competitive behavior when trading a single stock. The simulated market is very similar to real open-market trading because the simulated exchange matches buyers and sellers using an order book modeled on the mechanisms used in actual electronic stock exchanges. Agents not only see the sequence of completed trades as they occur but can also see the current bids and offers (including the corresponding prices and quantities) in the order book. Many agents in this domain predict by computing an estimate of the future trade price using time series methods; in some agents, this is the only source of information used from the environment. Kearns and Ortiz also examine participants in the 2003 competition and find that only 9 of the 14 autonomous agents participating used information from the order book in their strategy. The fact that the top agents were variants of the static order book imbalance (SOBI) strategy[3] suggests that order book strategies that incorporate additional information beyond the price series may be more robust to changes in the market environment such as variations in volatility and trend behavior.

Additional evidence of this is provided by Sherstov and Stone [2004] which present several price time-series based strategies. The paper presents a comparison of several automated stock trading agents with significantly different strategies. The first is a reinforcement learning based agent that decides its action based on the exponential average of previous prices and the last price seen. The degree of smoothing for the exponential average was chosen based on off-line training. This second and third agents examined the trend seen in the observations of recent trades. The market making agent entered trades in the direction of the current trend but limited its overall position size by simultaneously placing the corresponding exit order. The agent places its exit order taking into account the current order book price. In all three cases, the agents use the time series of recent trades to determine behavior and ignore other available information (which is somewhat limited in this domain) about the market. None of these strategies are able to compete against the SOBI strategy in terms of performance (particularly when measured by the Sharpe ratio[4]). Of course, manipulation of the market structure by individual agents can in some cases overcome this robustness.

[3]Strategic order book imbalance is a market-making strategy where an agent will add offers into the order book when one side (long or short) is significantly smaller than the other side. This is a trading strategy known to work in real world markets.

[4]The Sharpe ratio is a measure of reward-to-risk (higher is better). This is often used to compare performance of trading strategies in portfolio selection and other finance applications.

2.2.5 Electricity Market Price Prediction

Another application of price prediction involves pricing for electricity generation in the public utilities market. Contreras et al. [2003] use time series (e.g. ARIMA) methods to perform both "real-time" (1 hour) and "day ahead" (24 hours) future electricity price prediction. This domain differs significantly from the other applications discussed here in that the electricity markets possess strong multiple seasonality and calendar effects. Due to this feature, the authors find ARIMA models are particularly well suited due to the autoregressive aspect of the data. Some general periodic tendencies in the data facilitate the use of rules such as the following: electricity prices are low at night and high during the day, more electricity is used during week days than on weekends, and more electricity is used in the summer than in the winter. Also, unlike in goods markets, electricity produce must be immediately consumed or it is lost. Overall, the features of this application domain differ significantly from commodity and supply chain management markets.

2.2.6 River-flow and Effluent Density Prediction

The set of prediction methods popular for river-flow and effluent applications has undergone significant changes in recent years. The primary goal of research in prediction of river-flow is to precisely predict peak flows. This information is then used to determine flood stage (water height at a specific location) and to predict flow to facilitate efficient water usage. Before the 1980s, the most popular methods for this task were deterministic hydrologic models [Kitanidis and Bras, 1980]. These methods require significant domain expertise and the model parameters must be explicitly calculated for the particular target location. The advantage of such models is that the recomputing the output based on new input data is relatively inexpensive due to the deterministic nature of the model. As these models were unable to provide highly accurate predictions, computationally expensive methods began to be used that could capture greater granularity. Another disadvantage of deterministic models is that expensive reconstruction of the model using hydrologic principles is required when the physical characteristics of the river basin changes due to natural or human-initiated changes.

As research on artificial neural networks (ANNs) became popular in the 1990s, their application to river-flow prediction was studied [Li et al., 2003]. While application of ANNs to this domain was relatively straightforward, authors in [Imrie et al., 2000] highlight some critical weaknesses in the straightforward approach. First, ANN models cannot reliably generalize for input values outside the ranges seen the training data. Second, the structure (number of hidden layers, number of nodes) of the ANN model must be decided *a priori*. Without any

algorithmic guidance to determine network structure, it is possible the optimal network structure may be missed. Lastly, the method provides no implicit way of determining the fitness of individual inputs in the optimal feature set. The authors attempt to remedy the first problem by applying an iterative network construction strategy and a novel technique for training stopping criteria. The iterative construction method called the cascade-correlation learning architecture [Fahlman and Lebiere, 1990] starts with a simple network of one layer and adds randomly-weighted hidden nodes iteratively until the stopping criterion is reached. Leahy *et. al.* provide research that addresses the problem of determining input fitness [Leahy et al., 2008]. Their automated method performs complexity reduction through a genetic algorithm technique; specifically, the method works by randomly removing internal edges to determine a minimal structure that still provides a good prediction result. The above two methods outline two differing approaches to producing an ANN with minimal complexity to maximize generalization: the former uses iterative construction, while the latter uses a complexity reduction technique. Several techniques for reducing network complexity were pioneered outside the spatial domain using genetic algorithms [Sexton et al., 2004], optimal brain damage [Cun et al., 1990], and weight decay procedures [Weigend et al., 1990]. Overall, we consider the prediction task for a small target network with a small number (< 10) of gauge sites.

Commodity and Foreign Exchange Futures Market Prediction. Trading in commodity markets often involves using time series analysis to model changes in price. Barbosa and Belo [2008] show the promise of using time series models to optimally time entries and exits from the spot market in foreign currency trading. Kaufman [2005] provides an overview of the time-series, pattern recognition, and event-driven techniques popular in building commodity trading agents. Many of the techniques discussed rely on domain-specific features in the model formulation and are difficult to generalize to other domains.

2.2.7 Real Estate Price Prediction

Another application domain for prediction which possesses a potentially complex spatio-temporal component is real estate pricing. At first glance, real estate price modeling appears to be a challenging environment for prediction due to significant property heterogeneity and data sparsity. In spite of these challenges, several models have been developed for this domain [Pace et al., 1998]. Perhaps surprisingly, models in this domain are able to achieve prediction error rates of 10 to 15% RMSE when compared to actual sale prices. Models in this domain have the benefit of transaction data that is of consistently high data quality due to the needs of the stakeholders involved. Also, significant training data will exist for nearly any target region to be modeled. Cold starting can be a problem in this domain. A data bootstrapping process is needed for regions with little data (e.g. new development or areas with a slump in sales).

Pace et al. [2000] construct a regression model that achieves sufficient performance over 14 variables with as little as 160 training samples. This set of 14 features is obtained from a full set of 211 observable variables in their dataset. A parameter selection process examines the fitness of each input, and if relevant, it is added to the model. In feature-selection terminology, this is a filter-based feature selection approach.

2.3 Prediction Models

This section examines the range of prediction models used in the application domains discussed in the experiments of chapters 4-7.

2.3.1 Spatial-Temporal Methods

An area of prediction research most similar to the feature selection framework presented in this thesis is in the area of sensor network data analysis and prediction. Work by Rodrigues et al. [2008] introduces Online Divisive-Agglomerative Clustering (ODAC), a novel hierarchical clustering algorithm that computes a clustering tree of the variables in settings with a large number (hundreds) of variables. The algorithm is a divisive clustering process that builds a hierarchy from the set of data streams (variables) by considering similarity using a single pass. At the beginning, all variables are assigned to one cluster. When processing each cluster, there is a possible division of each cluster followed by a potential agglomeration of each pair of children based on statistical threshold criteria. The algorithm is favorably compared against k-means clustering for the same task and is found to be better at finding a high quality tree. The quality of the generated hierarchies from both ODAC and from k-means clustering are compared against the ground truth hierarchy for several synthetic datasets. The primary application of this work is for efficient on-line processing processing of large numbers of incoming data streams.

The problem of predicting the aggregate of several spatially correlated variables is addressed in [Giacomini and Granger, 2004]. Specifically, 4 alternative approaches are discussed each having a different the amount of spatial relationship information available to the model. These models are tested on synthetic data with spatial and temporal relationships specified a priori. The authors find that the Space-Time AR (autoregressive) time series model improves prediction accuracy especially in cases where the maximum amount of spatial correlation is bounded. Our proposed model framework's output can be encoded as a Space-Time AR time series model for spatial data domains.

A survey discussing the particular issues associated with many types of spatial data is available in [Shekhar et al., 2011]. In particular, the authors indicate that spatial relationships are often not explicitly stored during data collection. Unlike non-spatial relationships which are

often explicitly stored, it may be possible to infer non-spatial relationships from spatial data but information is often lost. Our proposed work seeks to improve on this problem by allowing some encoding of spatial information in model construction.

2.3.2 Multivariate Techniques

The literature contains several classes of methods that systematically leverage information from multiple related time series including vector autoregressive moving average (vector ARMA) and multivariate regression methods.

Vector ARMA is the multivariate analog of the ARMA time-series predictor presented in [Box et al., 1994; Tiao and Box, 1981]. Vector ARMA is applied in [Olason and Watt, 1986] using the MISO TFN algorithm (Multiple Input Single Output Transfer Function-Noise model) to improve river flow predictions on a network of hydropower generating stations. Given daily flow data from 3 upstream and one downstream reservoir, the authors developed an improved flow prediction model for both next-day and two day ahead predictions. The improvement was made using three years of daily flow measurements from all four sites and reduced standard error over the conventional approach of using separate time-series ARMA models for each site.

The vector ARMA also appears to be effective in domains with a small sample size. Disadvantages of vector ARMA stem from its sensitivity to collinearity in the input variables; the examples from the literature use only a small number (3-5) of input variables.

The family of related methods, multiple linear regression (MLR), principal component regression (PCR) and PLS regression are techniques first used with significant success in chemometrics, social science, and econometrics [Martens and Næs, 1992]. PLS regression is particularly successful on problems having a large number of highly collinear variables and a small number of samples [Wold et al., 1983].

Both PLS and PCR are types of bi-linear[5] calibration methods used for multivariate regression. Both also perform a mapping from the k-dimensional input vector X into an adjustable and lower-dimensional space $\hat{T}$ using a linear mapping $\hat{\mathbf{V}}$. Another set of loadings $\hat{\mathbf{P}}$ are used to map from the intermediate space $\hat{T}$ to the target scalar variable Y. In PCR, the mapping from the input features to the intermediate space with dimensionality α (the number of principal components to use in prediction, a parameter in the model) is performed using the principal components computed from the training set observations $\boldsymbol{X}$. The PCs can be computed iteratively using the NIPALS algorithm [Martens and Næs, 1992]. The first principal component represents the direction of maximal variance in the input set $\mathbf{X}$. When the contribution of this vector is removed, the second principal component represents the direction of maximal variance

[5]Bi-linear methods have two sets of linear mappings: the first is from a original space to a latent variable space, the second is from the latent variable space to the response space.

of the residual. This iteration is performed α times.

The loadings directions (e.g. the PCs) of the input matrix $\boldsymbol{X}$ used in PCR or PLS can be computed directly using eigendecomposition, singular value decomposition (SVD), probabilistically using maximum likelihood, or iteratively using the Non-linear Iterative PArtial Least Squares (NIPALS) approach Wold [1966]. NIPALS results in nearly identical results to SVD (within numerical precision). The advantage of the NIPALS approach is efficiency. There is no need to compute the full covariance matrix or inverse which are needed in the case of eigendecomposition. SVD requires a large convex optimization system to be solved as well. Maximum likelihood requires a full specification model (the stochastic properties of the residual must be computed) which is hard to model due to interdependencies between variables.

The disadvantage of PCR is that the variations captured by each PC may be irrelevant for predicting the variable of interest y. PLS is generally preferred over PCR because the computation of scores $\hat{t}_i$ in PLS takes into account variations in the y value, where as in PCR $\hat{t}_i$ is computed with no regard for the y values in the training set. The details of PLS are discussed in the next section.

2.3.3 Partial Least Squares Regression (PLS)

The methodology presented here does not make alterations to the core PLS algorithm, so our treatment of PLS will not be exhaustive. Several implementations of PLS exist [de Jong, 1993; Dayal and MacGregor, 1997; Martens and Næs, 1992, for example,]; each with its own performance characteristics. The implementations differ mostly in the way the scores (similar to the principle component directions in PCA terminology) are chosen: the score vectors can be orthogonal or not, normalized or not, etc.[6] This thesis uses the orthogonalized PLS with Non-Linear[7] Iterative Partial Least Squares of [Wold et al., 1983]. PLS was chosen over similar multivariate techniques including multiple linear regression, ridge regression [Hoerl and Kennard, 2000], and principal component regression [Jolliffe, 1982] because it produces generally equivalent or better performance than the others and has the ability to adjust model complexity. Specific advantages of this algorithm are presented.

Partial least squares regression is particularly applicable to modeling economic phenomena. First, PLS regression is able to handle very high-dimensionality inputs by performing an implicit

[6] It may be useful to have orthogonal scores vectors because factors are often independent in many application domains. This can alleviate model calibration difficulties due to many collinear variables Abdi [2010]. Alternatively, it may be useful to have orthogonal loadings for factor analysis. Finally, both can be orthogonal as in [Ergon, 2002].

[7] The NIPALS approach is called non-linear because it is a systematic method to linearize the estimation of a non-linear model. The non-linear model being linearized is usually a system of principal components but could be another representation such as canonical correlations. The NIPALS estimate is of the form $y_i = \sum_{a=1}^{k} \beta_{i,a} x_a + e$, for a k component model, where e is the model residual. At each of the k iterations $\sum_i {e_i}^2$ is minimizedWold [1966].

dimensionality reduction from the number of inputs to the number of PLS factors. Second, the model complexity can be adjusted by changing the number of PLS factors to use in computing the regression result. This value is adjusted in our experiments to determine the optimal model complexity in each prediction class. Third, the algorithm is generally robust to many collinear or irrelevant features.

Mathematically, PLS regression deterministically computes a linear function that maps a vector of the input features x_i into the output variable y_i (the label). Using a PLS regression model for a particular variable y requires a calibration to be performed over a set of training samples to determine the model as computed using Algorithm 1. The model can be used for prediction as described mathematically in Algorithm 2.

Algorithm 1: PLS1 (one response variable) Calibration

input : A matrix X containing n training samples, n rows each with m features. The corresponding vector y containing n labels for the training set. Model complexity A_{max} chosen so that $a = 1, \ldots, A_{max}$.
output: Loading arrays $\hat{W}$, $\hat{Q}$, and $\hat{P}$.

STEP 1 $X_0 = X - 1\bar{x}'$, where $\bar{x}'$ is a vector of the mean values of the variables in X.
$y_0 = y - 1\bar{y}$, where $\bar{y}$ is the mean value of y

optional Normalize columns in X_0 to have equivalent variance. (Divide each column by its variance.)

for $a = 1 \rightarrow A_{max}$ **do**

STEP 2.1 Using least squares, compute normalized local model $\hat{w}_a$
$\hat{w}_a = X'_{a-1} y_{a-1} / \|X'_{a-1} y_{a-1}\|$

STEP 2.2 Estimate scores $\hat{t}_a$ using model $\hat{w}_a$.
$\hat{t}_a = X_{a-1}\hat{w}_a$ (since $\hat{w}'_a \hat{w}_a = 1$)

STEP 2.3 Estimate x-loadings p_a using scores $\hat{t}_a$.
$\hat{p}'_a = X'_{a-1}\hat{t}_a / \hat{t}'_a \hat{t}_a$

STEP 2.4 Estimate y-loadings q_a using scores $\hat{t}_a$.
$y_{a-1} = \hat{t}_a q_a + f$
$\hat{q}_a = y'_{a-1}\hat{t}_a / \hat{t}'_a \hat{t}_a$

STEP 2.5 Update X and y with contribution of current a.
$X_a = X_{a-1} - \hat{t}_a \hat{p}'_a$
$y_a = y_{a-1} - \hat{t}_a \hat{q}_a$

An often ignored aspect of PLS and PCR methods is the ability to perform implicit error checking by examining the residual values from each prediction. Sample outliers are much more likely to have large magnitude residual values even if their y value maps to the generally observed range. When a sample with a large residual is found, the resulting prediction may have low confidence relative to other samples.

The behavior of PLS, MLR, and PCR regression techniques differ from the time-series based techniques due to the absence of a "memory" component. Time series models like ARMA can be affected by outliers in the input data, and the effect of an outlier can persist in the output prediction long after the outlier was first observed [Hillmer et al., 1983]. This is in contrast

Algorithm 2: PLS1 Prediction

input : Populated feature vector $\mathbf{x}_i$, calibration $\bar{x}$, calibration $\bar{y}$, loading weights $\hat{W}$, loadings $\hat{P}$, loadings $\hat{Q}$
output: Prediction of $\hat{y}_i$

STEP 1 Center observation of feature vector x_i.
$x_{i,0} = x_i - 1\bar{x}'$

optional Normalize variance $x'_{i,0}$ by dividing each value by its column variance in calibration X.

for $a = 1 \rightarrow A_{max}$ **do**

STEP 2.1 Compute contribution of $\hat{w}_a$ to y_i.
$\hat{t}_{i,a} = x'_{i,a} - 1\hat{w}_a$
$x_{i,a} = x_{i,a-1} - \hat{t}_{i,a}\hat{p}'_a$

STEP 3 Compute prediction of y_i
$\hat{y}_i = \bar{y} + \Sigma_{a=1}^{A} \hat{t}_{i,a}\hat{q}_a$

Alt. Alternative formulation in 1 step using $\hat{b}$.
$\hat{b}_0 = \bar{y} - \bar{x}'\hat{b}$
$\hat{B} = \hat{W}(\hat{P}'\hat{W})^{-1}\hat{Q}$
$\hat{y}_i = 1\hat{b}_0 + x'_i\hat{B}$

to the regression techniques: input observations that are outliers will only affect the output variable when the outlier is still in the input set. In adversarial settings like TAC SCM, where opponents have the ability to temporarily affect the input data to a prediction computation, the long term effect of outliers in time-series predictors is particularly undesirable.

2.4 Algorithm Parameter ("hyperparameter") Setting

A significant and under-explored area of machine learning is in principled techniques for setting algorithm parameters in model-based machine learning. This is commonly referred to as hyperparameter optimization. Many learning algorithms possess parameters which must be assigned a priori for the algorithm to be used. There are conventions for specific model parameters in many cases, but an automated exploration of the parameter space may be beneficial to maximize model performance. Analysis of the hyperparameter search may also be useful for determining a method's sensitivity to parameter choices. Automated model configuration forms an integral part of this thesis and is discussed extensively in Chapter 3.

At a high-level, there are several techniques from the literature. A common approach is to discretize the range of values for each parameter and exhaustively search all combinations; this is commonly known as a grid search [Larochelle et al., 2007]. Choosing the grid resolution for each parameter involves tradeoffs: too fine a resolution makes exhaustive solution infeasible and too coarse a resolution may cause a suitable configuration to be missed.

Non-exhaustive search schemes are applied to feature selection to ensure that a search can be performed even for very large search spaces. The non-exhaustive search schemes can also have an

additional benefit in preventing oversearching. In model-based machine learning, it is possible that searching a very large space of possible configurations can lead to poor generalization performance of the model. Quinlan and Cameron-Jones [1995] note that "expanding search leads eventually to a decline in predictive accuracy as idiosyncrasies of the training set are uncovered and exploited." In the experiments, the algorithm's search extent is varied by setting the beam width in a beam search (a memory-limited variant of best-first search). The oversearching problem is orthogonal to the problem of overtraining (increasing model precision and complexity to the extent that generalization performance on new samples is degraded) as oversearching can occur even for simple models.

Another approach is to sample randomly in the hyperparameter space to rapidly find good configurations [Bergstra and Bengio, 2012]. Sampling efficiency can be further improved by using a directed sampling process in which the model's performance is modeled as a black-box using a high-dimensional function approximation [Bergstra et al., 2011]. This response function approximation approach requires fewer configuration evaluations compared to other methods. It leverages the smoothness of the response surface to discover systematic trends which can prevent unneeded evaluations in uninteresting (i.e. not varying) regions of the response surface.

2.5 Model Testing Configuration – Experimental Protocol

When applying machine learning to time-series datasets, paying special attention to the method of dividing the data into training, testing, and validation sets is important for obtaining results that can generalize for novel, not yet observed, instances. Gama and Gaber [2007] provide an in-depth discussion of training set selection with an eye toward on-line learning scenarios where the concept to be learned may be changing over time. The choice of splitting approach can dramatically effect the prediction performance. For example, in a changing concept environment, the performance differences between a percentage split and cross-validation can be large. This is especially important in time series data and when lagged variables are used to predict trends. In such a dataset, adjacent samples are not independent and identically distributed (*i.i.d.*) which is a core assumption of many machine learning algorithms.

To address this topic further, we empirically evaluate several common approaches for cross validation for time-series data in Section 4.2.

2.6 Performance Measures and Comparisons

A wide range of statistical approaches exist for comparing prediction model performance in a regression setting. We examine several modern approaches and show the rationale behind the

methods used for our application domains.

An ubiquitous evaluation method for prediction models is in measuring accuracy. Specifically, many works minimize the mean error (distance between a sample's true value act_i and the model's prediction est_i for the sample) over all samples in some evaluation dataset (with n samples). It is also possible to normalize the errors so that each sample contributes equally which is the case with mean relative error.

$$\text{mean error} = \frac{1}{n}\sum_{i=1}^{n}|\text{est}_i - \text{act}_i| \tag{2.6}$$

$$\text{mean relative error} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\text{est}_i - \text{act}_i}{\text{act}_i}\right| \tag{2.7}$$

The above approach is good because as a statistic it is relatively robust to outliers, it is invariant to the length of the dataset, and it tries to balance errors over all samples. Often it is valuable to minimize large errors (outliers) by preferring methods that penalize large errors. This can be accomplished by using the root mean square error (RMSE) measure. It is also possible to normalize the statistic so that each sample contributes equally (rRMSE) even if the magnitude of the samples is very different.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{est}_i - \text{act}_i)^2} \tag{2.8}$$

$$\text{rRMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\text{est}_i - \text{act}_i}{\text{act}_i}\right)^2} \tag{2.9}$$

The accuracy measurement approach for performance evaluation is used in several of the application areas in this thesis. However, the ultimate purpose of many prediction tasks is to build accurate models for input into a decision-making processes. Another evaluation approach is to determine the cost of using a particular model for a particular decision process. By comparing the costs incurred from using different competing models, a best (i.e. least cost) model can be determined among the alternatives.

An absolute benchmark that can be computed in many cost-motivated domains is the expected value of perfect prediction (EVPP) measure Wellman et al. [2004]. Choosing the alternative that can achieve costs closest to the optimal decision (in hindsight) is desirable in many decision processes while accuracy is a secondary criteria. The cost measurement approach is taken for the airline ticket price prediction task presented in this work (Chapter 5) as well as in previous work in the domain Etzioni et al. [2003].

Another factor to consider in performance evaluation is the model complexity. Balancing error and model complexity can be achieved using methods such as AIC (Akaike Information Criterion, [Akaike, 1974]) or BIC (Bayesian Information Criterion, [Yang, 2005]). These methods balance model complexity (in terms of the number of free parameters, in linear models, or degrees of freedom, generally) against accuracy (deviation of the predictions from the true labels on the test set) on the test set. AIC is useful for comparing among several possible calibrated model choices using a single measure. BIC is a Bayesian approach and assumes the true model exists among the set of choices. Of course, if the true model does not exist or is not among the set of alternatives, the statistical results will not be correct.

In conclusion, there are a wide variety of performance measures that are applicable to prediction model evaluation. Invariably, the "best" model found will depend on the performance measure in many cases. It is critically important to determine a good evaluation criteria prior to embarking on prediction model comparison. While accuracy may seem appropriate for almost all domains, the most accurate model for a domain may not produce the best empirical decision outcomes. In the next section, we show the implementation-level details of the feature selection framework used in the experiments and formalize the discussion with precise mathematical notation.

Chapter 3

Methods

Making future time series predictions in a regression setting using information from recent observations is the focus of this thesis. The *supervised regression* setting we consider involves predicting a value for a continuous target variable using observations of other variables in the system. A regression model is useful for predicting ordinal, interval, or ratio variable types. This is in contrast to the *supervised classification* setting where the prediction involves assigning one of several (nominal) group labels to each sample.[1] In the classification case (with nominal type variables), there is often no meaningful order to the class labels, and the model needs only to be precise enough to obtain the correct label among the choices. Regression is often a harder learning task because the precision of the target value is important.

A major challenge common to both regression and classification is choosing a subset of variables to use in the model given a large number of possible variables in an application setting. At a high level, this problem of choosing the subset of variables to use as input to the learning algorithm is the problem of *feature selection*. This chapter proposes several search strategies for feature selection that can improve prediction model performance. This chapter discusses six methods for feature selection specifically relevant for regression domains with a time series component and known (possibly spatial) relationships between variables. Two fully automated methods are presented from the literature: CFS and BFS. Four methods utilizing domain knowledge as part of the Developer-Guided Feature Selection (DGFS) framework are presented as well: DGFS subtypes Exhaustive, Greedy, Random, and Guided. Using each of the proposed methods, we seek to build a regression model, a mathematical function that takes as input a set of observations from a set of independent variables and estimates the value of some

[1] The group assignments in classification tasks are usually disjoint, but there are many problem areas where multi-label classification is useful such as attributing musical sound clips to specific instruments [Xioufis et al., 2011].

dependent variable (target).

When calibrating such a supervised regression model, it can be beneficial to use many input variables to maximize prediction accuracy. Generally, adding more relevant variables to the feature set of a model to improve the accuracy of a prediction model seems to be an attractive idea. Observations from many environments are naturally noisy and the underlying generating function to be modeled is often complex, so having additional relevant inputs should reduce the uncertainty caused by noise. However, in practice, learning algorithms can perform well when the input data are limited to a small set of highly relevant variables. Adding more relevant variables can have diminishing returns and can even degrade performance. For example, if a group of variables is highly collinear (i.e. all pairs have high covariance), some learning algorithms will apportion equal importance to these. Each individual variable in this case may appear to not be very relevant because of the low importance assigned. Unintuitively, the addition of weakly relevant and even irrelevant variables can instead improve prediction performance on real data [John et al., 1994]. This can occur due to correlations that can occur between input variables. The feature selection process chosen can constrain or otherwise affect the outcome of the model selection and calibration. This is referred to as the feature selection bias. As these observations illustrate, feature selection is complex and involves tradeoffs.

Optimal model performance with respect to the learning algorithm, data set, and feature selection bias can be measured through the feature selection search methods in this chapter. Presenting a range of methods for determining a subset of input variables to achieve this ideal is the purpose of this chapter. The novel methods proposed serve to bias the feature selection by incorporating domain knowledge from the model developer (specifically knowledge of the relationships between variables).

We begin with a formal description of the feature selection task and terminology used in the feature selection domain. First, consider an idealized multivariate (multiple input variables) learning scenario with an unbounded (an unlimited number of samples can be observed) dataset drawn from the same underlying generating function. It is possible to classify each variable in the scenario into one of three types: strongly relevant, weakly relevant, or irrelevant [Kohavi and John, 1997]. A regression model can be constructed as a function of zero or more (and possibly all) of the variables. The best performing model in the model space for any subset of input variables is called the *optimal model* with respect to the learning algorithm and dataset. It is important to note that the optimal model may or may not be unique. There could be many different subsets of variables which satisfy this requirement based on the performance objective. A variable is *strongly relevant* if it is always present in each of the optimal models. A variable is *weakly relevant* if it is present in at least one of the optimal model equivalence class. The weakly relevant variable may be among a group of variables that are correlated but are not

uniquely informative so are interchangeable. Finally, an *irrelevant variable* is one that is not present in any of the optimal models.

In non-idealized scenarios where noise is present and the amount of data is not unbounded, it is possible (and likely) that conceptually irrelevant[2] variables can be found to be useful for prediction (and therefore relevant). When the number of observations in the training set is very limited and the number of variables is large, irrelevant variables can appear to be very informative due to random chance which induces spurious correlations. Such relationships do not generalize. Learning scenarios with many irrelevant variables make the calibration task difficult, and might make the learning algorithm likely to include variables that provide no predictive information when applied to new data. Feature selection can reduce the likelihood that ephemeral (e.g. highly irregular and unexpected) relationships will be used in the model. This is the argument in favor of using feature selection as a preprocessing step.

Next we will formalize our terminology for learning and feature selection with some precise notation.

3.1 Notation

The task we study in this chapter is supervised learning. We are given a data set D which contains a set of observations containing values of a consistent set of m features (independent variables): $\boldsymbol{X} = \{X^{(1)}, X^{(2)}, \ldots, X^{(m)}\}$. Each element in the dataset D is a tuple $\langle \boldsymbol{X}, Y \rangle$ where Y is the value of the dependent variable (target) for the corresponding observation $\boldsymbol{X}$. Each instance $\boldsymbol{X}$ is an element in the feature space $V^{(1)} \times V^{(2)} \times \ldots \times V^{(m)}$, where $V^{(i)}$ is the domain of the ith feature. In our experiments, we consider datasets with $V^{(i)}$ in $\mathbb{R}$ or $\mathbb{N}$, but in principle there is no restriction.

Formally, the algorithm calibrates through induction $\mathcal{I}$ a model M using dataset D: $M = \mathcal{I}(D)$. It builds a function M which can predict a label $\hat{Y}$ for an instance $\boldsymbol{X}$: $\hat{Y} = M(\boldsymbol{X})$. The model can generalize for instances of $\boldsymbol{X}$ not observed in D (i.e. the incoming instance to be labeled may never have been previously observed). The above supervised learning scenario involves learning over the set of all known features from the dataset D but a model is often built using an altered dataset as well $D_{S'}$ which may include both time delayed variables and a subset of the original variables.

We denote this set as S_D which is the feature set having all variables: $S_D = \{V^{(1)}, V^{(2)}, \ldots, V^{(m)}\}$. It is also valuable to calibrate a model using a subset of the m features: $S'_D \subseteq S_D$. A new dataset formed using a subset of features is called an *altered dataset* $D_{S'}$ and is made of instances of

[2]Conceptually irrelevant variables have no relationship to the target variable in the generating function (i.e. the variable does not appear in the generating function for the target). If the underlying generating function is known such as in the case of the synthetic data used in this chapter, it is possible to determine this.

$\langle \boldsymbol{X}_{S'_D}, Y \rangle$ where $\boldsymbol{X}_{S'_D} = \{X^{(i)} | V^{(i)} \in S'_D\}$.

The variables used in the model can be classified as strongly relevant, weakly relevant or irrelevant.

Definition 1 (Strongly relevant feature)**.** A feature is strongly relevant if and only if removing it from the optimal feature set[3] will decrease the model's performance (Equation 3.1).

$$p(Y = y | X^{(i)} = x, S^i = s^i) \neq p(Y = y | S^{(i)} = s^i) \tag{3.1}$$

$$S^{(i)} = \{X^{(1)}, \ldots, X^{(i-1)}, X^{(i+1)}, \ldots, X^{(m)}\} , \tag{3.2}$$

where $S^{(i)}$ is the set of all features except $X^{(i)}$ (Equation 3.2). $x^{(i)}$ denotes a value assignment of feature i. Similarly, $s^{(i)}$ denotes an assignment of values to all features in $S^{(i)}$.

Definition 2 (Weakly relevant feature)**.** A feature is weakly relevant if it is not strongly relevant and it can sometimes contribute to model performance (Equation 3.3).

Weakly relevant variables will appear in one or more of the optimal feature sets.

$$p(Y = y | X^{(i)} = x, S'^i = s'^i) \neq p(Y = y | S'^i = s'^i) , \tag{3.3}$$

where $S'^{(i)}$ is a subset of $S^{(i)}$.

Definition 3 (Irrelevant feature)**.** A feature is irrelevant if it is not strongly or weakly relevant (i.e. it can never improve prediction accuracy).

In idealized scenarios, it is possible for irrelevant variables to exist. But in models built on real data (with limited size datasets and noise) conceptually irrelevant variables can be significantly correlated with other variables due to random chance or due to properties of the generating function. This can cause conceptually irrelevant variables to improve predictions.

The process of feature selection determines a subset S'_D of the original features S_D in the dataset D. The objective is usually to maximize some objective function (e.g. accuracy) of a model M. The final score reported is usually the mean of the error when predicting on a hold-out set used for testing. This forms an estimate of accuracy on future (unknown) observations. Fully automated feature selection methods use only the dataset (and possibly the learning algorithm) to determine a good feature subset.

In temporal datasets, it may be useful to add time delayed observations of variables to the feature set provided to the learning algorithm. In this type of domain, the dataset is composed

[3]The optimal feature set is optimal (no better performing set can be obtained) with respect to the algorithm, dataset, and feature selection strategy employed.

of (one or more) observation sequences: $D = \langle \ldots, \{x^{(1)}(t-1)\}, \{x^{(1)}(t)\}, \{x^{(1)}(t+1)\}, \ldots \rangle$. Time delayed observations can be added to the feature set by employing the unit delay operator ϕ_{-1} which delays the output by 1 time unit (i.e. $x(t-1) = \phi_{-1}x(t)$). In the case of an univariate time series (one variable only) a simple time series linear regression model can be encoded using the time delay operator as shown in Figure 3.1 [Wan, 1994]. A set of permissible time lags is assigned to each feature class as well: $l^{(i)} = \{\{a\}, \{b\}, \ldots\}$ where $l^{(i)}$ is the set of lag configurations for feature class i. The set of time lags assigned to all feature classes in the domain is $\boldsymbol{L} = \{l^{(1)}, l^{(2)}, \ldots, l^{(q)}\}$.

A model developer can incorporate additional knowledge into the feature selection process under the DGFS framework. Additional notation is necessary to describe this process precisely. For a dataset D with feature set S, the developer describes a set of q feature classes $\boldsymbol{F} = \{f^{(1)}, f^{(2)}, \ldots, f^{(q)}\}$. Each feature class is composed of a subset of features specified by the user: $f^{(i)} = \{X^{(j)}, X^{(k)}, \ldots\}$. In the experiments, the assignments of variables to feature classes are disjoint[4] but in principle this is not a necessary restriction.

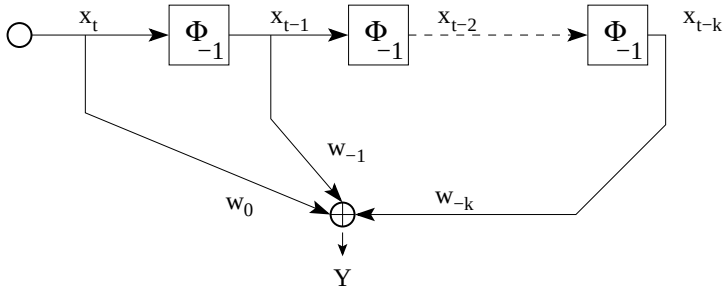


Figure 3.1: Time delay operator for time series data. w_{-i} refers to the weight of the ith offset in the model M (where $M : \{X_t, \ldots, X_{t-k}\} \longrightarrow Y$).

Also, the model developer can specify a set of feature class constraints: $\boldsymbol{C} = \{\ldots, (f^{(i)} \subset f^{(j)}), (f^{(k)} \subset f^{(l)}), \ldots\}$, where $(f^{(i)} \subset f^{(j)})$ means that for feature class $f^{(i)}$ to be included in the configuration $f^{(j)}$ must be included in the configuration as well. These constraints bias the feature selection process further and avoid exploration of large areas of the configuration space. Each constraint tuple defines a relationship between two feature classes and the operator specifies the mathematical property to satisfy. The constraint operators will depend on the domain context, in these experiments only mathematical comparisons $(<, >, \leq, \geq)$ and set operations $(\subset, \subseteq)$ were used. In non-temporal data sets, time lags have no conceptual meaning. As there is no logical reason to combine information from adjacent samples, time lag assignments for each feature class in non-temporal data are zero offset or empty (no assignment): $l^{(i)} = \{\{0\}, \varnothing\}$. The expressiveness of the constraint operators could be extended as necessary for future application

[4]Each variable is assigned to at most one feature class.

domains.

An augmented dataset can be generated from a dataset, a feature class, and a set of feature class time lag assignments using the feature set generator "`Aug`": $D_{\text{Aug}} = \texttt{Aug}(D, L, F, CL)$. The augmented dataset must satisfy all constraints in the constraint set CL to be a valid dataset. A small worked example in Figure 3.2 shows the five augmented datasets which can be generated from the original dataset, the feature class lag assignments, the feature classes, and feature class constraints.

Configuration

$$D = [\ldots, \langle\{x_1, x_2, x_3, x_4, x_5\}, y\rangle, \ldots]$$
$$F = \{f^{(1)}, f^{(2)}, f^{(3)}\}$$
$$L = \{l^{(1)}, l^{(2)}, l^{(3)}\}$$
$$l^{(1)} = \{\{-8\}\} \quad l^{(2)} = \{\{0\}, \varnothing\} \quad l^{(3)} = \{\{0, -1\}, \{0\}, \{-1\}\}$$
$$S^{(1)} = \{X_1, X_2\} \quad S^{(2)} = \{X_3\} \quad S^{(3)} = \{X_5\}$$

Yields 5 possible augmented datasets due to the alternative time lag assignments in each feature class.

$$|l^{(1)}| \times |l^{(1)}| \times |l^{(1)}| - \langle\text{constraint exclusions}\rangle = 1 \times 2 \times 3 - 1 = 5$$

Set of altered datasets

$$D_{\text{Aug1}} = [\ldots, \langle\{x_1(-8), x_2(-8), x_3(0), x_1(0), x_1(-1)\}, y\rangle, \ldots]$$
$$D_{\text{Aug2}} = [\ldots, \langle\{x_1(-8), x_2(-8), x_3(0), x_1(0)\}, y\rangle, \ldots]$$
$$D_{\text{Aug3}} = [\ldots, \langle\{x_1(-8), x_2(-8), x_3(0), x_1(-1)\}, y\rangle, \ldots]$$
$$D_{\text{Aug4}} = [\ldots, \langle\{x_1(-8), x_2(-8), x_1(0), x_1(-1)\}, y\rangle, \ldots]$$
$$D_{\text{Aug5}} = [\ldots, \langle\{x_1(-8), x_2(-8), x_3(0), x_1(0)\}, y\rangle, \ldots]$$

Figure 3.2: Example of augmented dataset generation

Such altered datasets ($\{D_{\text{alt1}}, \ldots, D_{\text{alt5}}\}$) can be used directly with a learning algorithm and will incorporate the intended feature subset without any modification to the learning algorithm. This facilitates experimentation using existing algorithms with no knowledge of the feature selection process and is a distinct advantage of a wrapper-based approach (i.e. the learning method can be treated as a black box).

In machine learning experiments, the initial dataset D is typically split into several (usually disjoint) sets called training (D_{TR}), validation (D_{VA}), and test (D_{TE}) sets. The purpose of the training set is to calibrate the learning algorithm, the validation set is used as a hold-out set for the training phase to tune high-level hyperparameters of the learning or feature selection algorithms, and the test set is used as a final hold-out set to estimate the performance of the

model on future (unseen) samples.

Specifically, we maximize prediction accuracy on the validation set using a model calibrated on the training set.

3.2 Data Driven Feature Selection

Data driven feature selection methods use just the dataset and possibly the target learning algorithm (in the case of wrapper methods) to determine a relevant feature subset. Much of the work in feature selection concerns classification in domains with binary variables (for examples, see [Cardie, 1993; Kononenko, 1994]). Because our target applications are fundamentally regression problems, not all of the previous work is directly applicable. We describe the methodology of two common methods for automated feature selection from the literature that are applicable to multivariate regression domains.

3.2.1 Correlation-Based Feature Selection (CFS)

Correlation-Based Feature Selection (CFS), a filter-based method, considers only the incoming data to determine a good feature subset. In [Hall, 2000], the author succinctly states the core principle of the CFS algorithm: "Good feature subsets contain features highly correlated with the class, yet uncorrelated with each other."

CFS uses a best-first search procedure with an initial state containing the empty feature set (in the forward induction case). At each iteration, a set of candidate configurations is generated. Each new candidate is the current state with one feature class-time lag pair added. The candidates have an *edit distance*[5] of one from the current state. The merit heuristic for each candidate is computed (Equation 3.4), and the candidate is added to the set of candidate states. The unvisited candidate state with the highest merit heuristic value is used as the state for the next iteration. An additional stopping criterion is added to the best-first search as well: if the last 5 consecutive states expanded result in no improvement of the "best" observed merit value, then stop. The merit heuristic formula is:

$$\text{merit}_S = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}} \tag{3.4}$$

[5]Edit distance is a measure of displacement between two sets (a and b). It is the number of membership changes (adding or removing elements) that must be made to set a to translate it into set b.

where $\overline{r_{cf}}$ is the mean target-to-feature intercorrelation, $\overline{r_{ff}}$ is the mean feature-to-feature intercorrelation. The intercorrelation is simply the Pearson's correlation coefficient:

$$r_{xy} = \rho_{xy} = Corr(x, y) = \frac{Cov(x, y)}{\rho_x \rho_y} = \frac{E[(x - \mu_x)(y - \mu_y)]}{E[(x - \mu_x)^2]E[(y - \mu_y)^2]}. \tag{3.5}$$

The CFS method was devised for the classification case of supervised learning in which only categorical features and class labels are used. In instances of continuous features (e.g. real valued or integer variables), supervised discretization is performed to map all variables to a discrete categorical set. CFS uses the maximum description length principle for performing an entropy minimizing discretization using the method in [Fayyad and Irani, 1993]. The supervised discretization of all non-categorical features (and the target variable) may be a source of poor performance in regression settings involving many continuous features. The ordinal properties and small changes in the values of these features will be ignored when making feature correlation comparisons due to this aspect.

At a high level CFS selects a core of relevant features with little redundancy with the objective of minimizing groups of highly correlated variables. This makes CFS good for pruning large numbers of highly collinear variables.

3.2.2 Best-First Feature Selection (BFS)

Another common algorithm, Best-First Feature Selection (BFS), uses an in-situ approach to determine the best feature subset [Kohavi and John, 1997]. It evaluates many possible subsets and computes the prediction performance of each. It is called the wrapper approach because the underlying machine learning algorithm is used to do the scoring.

Algorithm 3: Best-First Search for Feature Selection ([Kohavi and John, 1997])

Data: feature set S, dataset D, feature configuration evaluation function $f(\dot{})$, minimum improvement increment ϵ
Result: best observed feature subset (best)

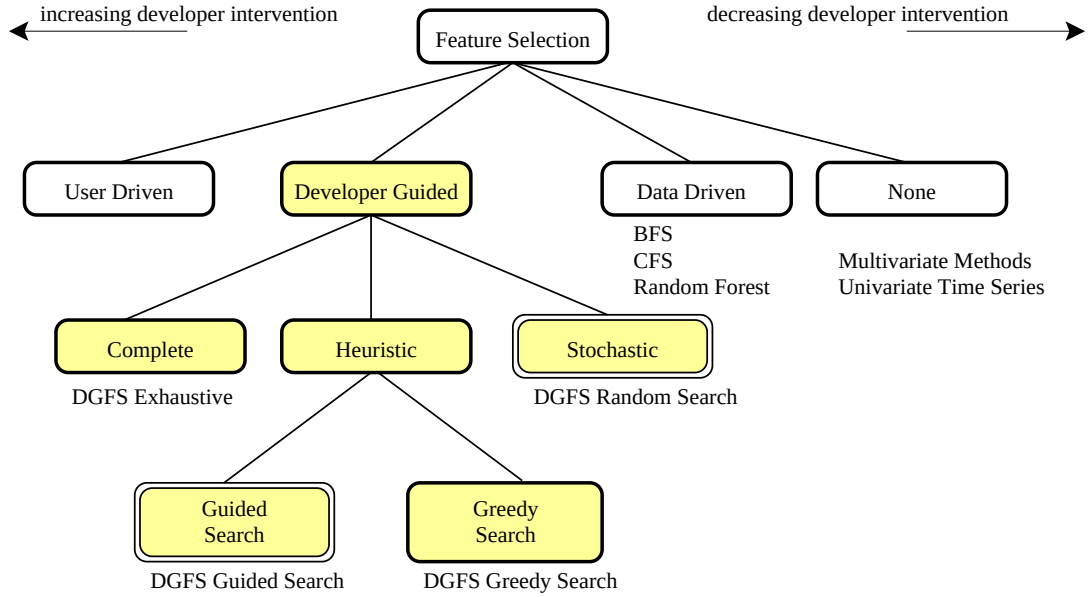
```
1  { OPEN ⟵ {∅},  CLOSED ⟵ {},  best ⟵ null}
2  while OPEN ≠ null do
3      state ⟵ arg min_{w∈OPEN} f(w, D)        /* pick best scoring configuration in OPEN  */
4      OPEN ⟵ OPEN \ {state}                    /* remove configuration state from OPEN  */
5      CLOSED ⟵ CLOSED ∪ {state}                /* add configuration state to CLOSED  */
6      if f(state, D) − ε < f(best, D) then
7          best ⟵ state
8      for V_i ∈ S do
9          child ⟵ state ∪ {V_i}
10         if (child ∉ OPEN) ∧ (child ∉ CLOSED) then
11             OPEN ⟵ OPEN ∪ {child}
12 return best
```

Algorithm 3 provides a forward induction version of the algorithm of [Kohavi and John, 1997], but it can be trivially modified to operate in a backward elimination mode as well (by changing the initial state and subset addition step). $f(w, D)$ is the scoring ("loss") function and computes the fitness of a feature set w on the validation set D_{VA} after calibrating on a training set D_{TR}.

3.3 Developer-Guided Feature Selection (DGFS)

Developer-Guided Feature Selection is an umbrella of methods which we propose that uses domain knowledge from the model developer to inform the feature selection search. Given a dataset D, lag set L, feature class membership mapping F, and (optional) constraint set CL, we devise a biased feature set search. The actual search process can be complete ("Exhaustive") or incomplete. The incomplete search can be directed ("Guided", "Greedy") or stochastic ("Random"). Each type of search is presented here, and the tradeoffs are discussed. DGFS is situated in the middle of the feature selection spectrum. It includes some automation while incorporating domain knowledge in the process (Figure 3.3).

Figure 3.3: Feature Selection techniques hierarchy. White nodes indicate techniques from previous literature (User Driven, Data Driven, None). Yellow-shaded nodes indicate DGFS-based approaches (Developer-Guided). Yellow-shaded double circle nodes nodes (Random Search, Guided) indicate DGFS-based feature selection extended using hyperparameter optimization to increase the efficiency of feature selection search.



Parameter optimization in model-based machine learning refers to the process of determining the best set of parameter values to use in the chosen model. Hyperparameters refer to the parameters of the learning algorithm that effect the outcome of the model calibration process but are not determined in the learning process (i.e. parameters are set *a priori*). For example, traditional neural network learning calibration parameters include the learning rate, the choice of activation function, the network layout of hidden nodes, and the number of hidden layers. Feature subset selection can also be considered as part of the hyperparameter selection process as well. Learning algorithm parameter choices are important because bad hyperparameter choices can cause bad model calibration results.

For this reason, choosing learning method parameters is an important topic in machine learning that is, at times, underexamined. While some learning algorithms possess theoretical guarantees on performance, in practice, the empirical performance observed through experimentation is valuable and is common practice as a criterion for model, algorithm, and hyperparameter selection. In literature presenting novel learning algorithms, hyperparameter selection encompasses a range of techniques:

- Some choices are based on convention.
- Some are based on expert knowledge of the domain. There may be a strong theoretical basis for a specific value for the given application.
- Some choices are based on experimentation.

The last choice is of particular interest because it is repeatable and can be applied in a principled way to new datasets or domains. We explore this approach by proposing four search methods.

3.3.1 DGFS – Exhaustive Search

A common approach to experimental optimization is to perform a complete uniform sampling of all combinations of parameter values. This is commonly known as a grid search or exhaustive search [Larochelle et al., 2007].

The exhaustive search mode of DGFS employs a complete enumeration of all valid feature subsets to determine a result (Algorithm 4). For each configuration, an evaluation is made by constructing the augmented training and validation datasets. The augmented training set is used to calibrate the learning model which is then scored using the validation set. The configuration with the best observed score on the validation set is the one used for the final evaluation on the hold-out test set. This final evaluation determines the error score for the experiment.

In the exhaustive search, we treat the feature class lag setting process as an outer-loop parameter optimization of the q feature classes. The optimization process must make one time

Algorithm 4: DGFS Exhaustive Search ([Groves and Gini, 2015])

Data: feature set S, dataset D, feature class specification F, list of time lag assignments for each feature class L, constraint set CL, feature configuration evaluation function $\mathtt{f}(\bullet, \bullet, \bullet)$
Result: best observed feature subset ($\boldsymbol{w}_{\mathsf{best}}$)

```
{ CONFIGLIST ⟵ {{}},   w_best ⟵ null}
build set of all possible valid configurations
for f^(i) ∈ F do
    NEXTCONFIGLIST ⟵ {}
    for w ∈ CONFIGLIST do
        for lagassignment ∈ L^(i) do
            NEXTCONFIGLIST ⟵ NEXTCONFIGLIST ∪ {w ∪ ⟨f^(i), lagassignment⟩}
    CONFIGLIST ⟵ NEXTCONFIGLIST
VALIDCONFIGLIST ⟵ {}
for w ∈ CONFIGLIST do
    if satisfiesConstraints(w, CL) then
        VALIDCONFIGLIST ⟵ VALIDCONFIGLIST ∪ w
evaluate each configuration to find best
for w ∈ VALIDCONFIGLIST do
    if (w_best = null) ∨ (f(w, D, F) < f(w_best, D, F)) then
        w_best ⟵ w
return w_best
```

lag set assignment for each feature class according to the permissible time lags: $L = \{l^{(1)}, ..., l^{(q)}\}$. The number of possible configurations (assuming no constraints) is $\prod_{i=1}^{q} |l^{(i)}|$. This quantity can be too large to search exhaustively when individual feature classes have many lags (i.e. when $|l^{(i)}|$ is large) or when there are many feature classes. For the exhaustive mode to be feasible, the set of possible lags for each feature class must not be too large. This problem can be handled either by possibly adding additional constraints to the feature classes or by using a non-exhaustive search process as explained in the following two subsections.

The advantages of an exhaustive search are in its simplicity, its complete coverage of all value combinations specified, and in the ability to analyze relationships between combinations of parameters. Unfortunately, it is not sample efficient[6] when compared to other methods: from our experiments it is approximately 10 times or 100 times less efficient. Also, many evaluations at high resolution (i.e. number of distinct values to evaluate for each hyperparameter is high) are necessary to achieve good coverage. A search which is too coarse can easily miss good configurations. The search for improved hyperparameter configurations can be made more efficient at the expense of completeness.

Algorithm 5: DGFS Greedy Search

Data: feature set S, dataset D, feature class specification F, list of time lag assignments for each feature class L, constraint set CL, feature configuration evaluation function $f(\bullet, \bullet, \bullet)$
Result: best observed feature subset ($\boldsymbol{w}_{\mathsf{best}}$)

```
{ F_remaining ⟵ F,   w_best ⟵ null,   w_state ⟵ {} }
while F_remaining ≠ {} do
    find all possible assignment successors from the state
    CHILDSTATELIST ⟵ {}
    for f^(i) ∈ F_remaining do
        for lagl ∈ l^(i) do
            possiblechild ⟵ w_state ∪ {⟨f^(i), lagl⟩}
            if satisfiesConstraints(possiblechild, CL) then
                CHILDSTATELIST ⟵ CHILDSTATELIST ∪ {possiblechild}
    find best successor to the current state for next iteration
    { w_bestchild ⟵ null,   f_remove ⟵ null}
    for ⟨w_childstate, f^(i)⟩ ∈ CHILDSTATELIST do
        if (f(w_childstate, D, F) < f(w_bestchild, D, F)) ∨ (w_bestchild = null) then
            w_bestchild ⟵ w_childstate
            f_remove ⟵ f^(i)
    w_state ⟵ w_bestchild
    F_remaining ⟵ F_remaining \ {f_remove}
    if (w_best = null) ∨ (f(w_bestchild, D, F) < f(w_best, D, F)) then
        w_best ⟵ w_bestchild
return w_best
```

3.3.2 DGFS – Greedy Search

The greedy search mode (Algorithm 5) is an incremental process that begins with the empty set for the initial state (i.e. an empty subset of feature classes). At each iteration, the algorithm evaluates each newly discovered configuration which is created by the addition of one feature class-time lag pair to the current subset. Unseen configurations violating the constraint set are discarded. Of these new configuration evaluations, the next feature class subset chosen as the state is the subset that had the best evaluation score. Once a feature class is assigned in the current state, its assignment is not evaluated again. This iteration continues until all feature classes have been added once to the in-progress state.

The greedy search has a worst case time complexity of $O(q * b/2)$ where q is the number of feature classes and b is the average number of possible lag assignments in each feature class. This search process is a kind of best first search (the best observed configuration is chosen at each iteration) and is greedy (no revision of the search is possible once a step is taken). The advantages of this approach are that it is intuitively simple and the total evaluation cost is bounded based on the input size. However, this greedy algorithm is sensitive to local minima

[6]Sample efficiency refers to the number of evaluations required to achieve a specific level of performance from the learning model.

in the response surface. The next two approaches overcome this disadvantage and can be even more efficient.

3.3.3 DGFS – Random Search

A random exploration of the configuration space does not, at first, seem to be a systematic approach. However, in practice, the efficiency of the random search is an improvement over a systematic-but-too-coarse search and is competitive with other approaches. A random search can be employed that is empirically very efficient [Bergstra and Bengio, 2012]. This algorithm samples from the hyperparameter space while obeying the sampling distribution (the "prior") specified for each parameter. Usually we sample from a set of discrete values, but we may also be sampling from a continuous distribution for some parameters. Repeated sampling in the parameter space and evaluation of candidate configurations continues until some stopping criterion is reached such as the number of evaluations, wall time, or a sufficient evaluation score is achieved. Usually, the best observed hyperparameter configuration (i.e. the configuration corresponding to the minimum observed value from all evaluations of the loss function) is given as the final configuration from the optimization process.

Algorithm 6: Sequential Model-Based Optimization (SMBO) [Hutter et al., 2011] used by DGFS Random search and Guided search

Data: feature set S, dataset D, feature class specification F, list of time lag assignments for each feature class L, constraint set CL, feature configuration evaluation function $f(\bullet, \bullet, \bullet)$, number of initial bootstrap samples N, $\mathcal{S}$ directed sampling function, $\mathcal{R}$ random sampling function
Result: best observed feature subset ($\boldsymbol{w}_{\mathsf{best}}$)

$\{\ \mathsf{score}_{\mathsf{best}} \longleftarrow null, \quad M_0 \longleftarrow \{\}\}$
$\mathsf{count}_{\mathsf{iterations}} \longleftarrow 1$
while *not* `stoppingCriteriaReached`$(\mathsf{count}_{\mathsf{iterations}}, \mathsf{score}_{\mathsf{best}})$ **do**
  if $\mathsf{count}_{\mathsf{iterations}} < N$ **then**
    $\boldsymbol{w} \longleftarrow \mathcal{R}(null, F, L)$ *// note: for DGFS Random search $n = \infty$*
  else
    $\boldsymbol{w} \longleftarrow \mathcal{S}(M_{i-1}, F, L)$ *// pick a new configuration to evaluate based on response model*
  $\mathsf{score} \longleftarrow f(\boldsymbol{w}, D, F)$
  if $(\mathsf{score}_{\mathsf{best}} = null) \vee (\mathsf{score} < \mathsf{score}_{\mathsf{best}})$ **then**
    $\boldsymbol{w}_{\mathsf{best}} \longleftarrow \boldsymbol{w}$
    $\mathsf{score}_{\mathsf{best}} \longleftarrow \mathsf{score}$
  $M_i \longleftarrow M_{i-1} \cup \{\langle \boldsymbol{w}, \mathsf{score} \rangle\}$
  $\mathsf{count}_{\mathsf{iterations}} \longleftarrow \mathsf{count}_{\mathsf{iterations}} + 1$
return $\mathsf{score}_{\mathsf{best}}$

An algorithm listing of the optimization process is given in Algorithm 6 and the sampling function used to determine the next configuration to evaluate is given in Algorithm 7 (for random sampling).

On why random sampling works. In practice, hyperparameter optimization explores a

Algorithm 7: Random Sampling Surrogate Function – $\mathcal{R}(\bullet, F, L)$

Data: feature class specification F, list of time lag assignments for each feature class L
Result: configuration to sample ($\boldsymbol{w}$)

```
{ F_remaining ⟵ F,   w ⟵ {} }
while F_remaining ≠ {} do
    for f^(i) ∈ F_remaining do
        sample a possible candidate lag for the feature class
        lagval ⟵ getRandomSetting(l^(i))
        w^(i) ⟵ lagval
    F_remaining ⟵ F_remaining \ {f^(i)}
return w
```

search space with low effective dimension (i.e. the problem can be mapped into a lower-dimensional space with little loss of information). By sampling randomly, more unique samples are taken from this lower dimensional space. Random hyperparameter search is good because it is more sample efficient than exhaustive and it is not sensitive to local minima in the loss function. Unfortunately, it may not find the global optimum if the number of samples is insufficient. While a random search does have attractive properties and is more efficient than an exhaustive search, a directed search can further increase sampling efficiency as we see in the next section.

3.3.4 DGFS – Guided Search

An alternative approach to the exhaustive, greedy, and random search methods is instead to direct the search using a model of the evaluation function ("loss") made with an approximation function. Modeling the evaluation function is valuable when computing the evaluation function is very costly. This approach allows for the number of evaluations to be minimized. If the response surface is locally smooth (i.e. small changes in the configuration only result in small changes to the evaluation score), such an approximation (which implicitly incorporates trends and covariances between pairs of hyperparameters) facilitates rapid discovery of the global optimal configuration.

Guided search uses the same overall evaluation process as Random search. This method is called Sequential Model-Based Optimization and is given in Algorithm 6 of the previous section. Guided search differs from Random search in that the sampling function incorporates a model of the response surface. The model is bootstrapped using an initial set of samples found using random sampling. The benefit of this estimation process through shrinkage is that it implicitly models uncertainty in unexplored regions and performs smoothing in explored areas. The sampling function $\mathcal{R}(\bullet, \bullet, \bullet)$ is replaced by $\mathcal{S}(\bullet, \bullet, \bullet)$ after the initial bootstrap. The Function "$\mathcal{S}$" (Algorithms 8-10) efficiently approximates the response surface for as-yet unevaluated configurations. It is used to direct the sampling process by balancing both exploration of the parameter

space and exploitation of information from previous samples. This hyperparameter search process is examined in detail for a variety of machine learning algorithms and benchmark datasets in [Bergstra et al., 2011]. The authors find that the approximation function chosen should be both locally smooth (to allow for estimation based on the set of noisy samples observed so far), should implicitly model uncertainty, and allow for incremental changes (for adding new observations) without complete reevaluation of the model. The Tree of Parzen Estimators (TPE) method is employed because of these desirable properties [Parzen, 1962].

The TPE method for hyperparameter optimization models the responses observed on a per-parameter basis using a Parzen estimator for each feature class (hyperparameter). A set of randomly sampled parameter values for each feature class is evaluated to determine the most likely well-performing parameter setting for each set (Algorithm 10, line 3). The configuration returned is a candidate likely to perform well based on the response estimates for each hyperparameter.

Algorithm 8: Tree of Parzen Estimators Sampling [Bergstra and Bengio, 2012] — $\mathcal{S}(M, F, L)$

Data: observed response samples M, feature class specification F, list of time lag assignments for each feature class L, number of candidates to sample for each parameter nc
Result: a promising configuration to sample (w_{best})

```
1  {LC ⟵ {},   w ⟵ {} }
2  // build lag candidates for each feature class by sampling a set of possible candidate lags for each
3  for f^(i) ∈ F do
4      for j ⟵ 1 to nc do
5          lagval ⟵ getRandomSetting(l^(i))
6          LC^(i) ⟵ LC^(i) ∪ {lagval}
7  // pick a new configuration using the settings that yielded the best Parzen estimate in each feature class
8  for f^(i) ∈ F do
9      build surrogate model for f^(i) given all observations so far
10     { Q_greater ⟵ {},   Q_less ⟵ {},   y_γ ⟵ getScorePrecentile(M, γ)}
11     for ⟨x, y⟩ ∈ M do
12         if y < y_γ then
13             Q_less ⟵ Q_less ∪ {⟨x_i, y⟩}
14         else
15             Q_greater ⟵ Q_greater ∪ {⟨x_i, y⟩}
16         L ⟵ buildGaussianProcess(Q_less)
17         G ⟵ buildGaussianProcess(Q_greater)
18         {p_best ⟵ 1.0,   bestlag ⟵ null}
19         find candidate lag set that probabilistically provides best improvement
20         for lag ∈ LC^(i) do
21             p ⟵ estimateScore(Q_greater, lag) ÷ estimateScore(Q_less, lag)
22             if lag_best = null or p_best > p then
23                 {lag_best ⟵ lag,   p_best ⟵ p }
24         w^(i) ⟵ lag_best
25 return w
```

The Parzen estimator (Algorithm 8, lines 16-31) is a mathematical model for estimating

black-box functions using pairs of Gaussian Processes. The estimation of the function provides probabilistic guarantees (bounds) on estimate accuracy based on all samples observed so far. As the number of samples goes to infinity, the estimate converges to the true distribution through shrinkage. This convergence to the true value is achieved by maintaining two stochastic processes that estimate a lower bound and an upper bound on a specific percentile (γ) of the response function across the input space. The Parzen estimator splits the samples into two disjoint sets based on the percentile threshold. The probability density function for a parameter setting x conditioned on a loss response value y is estimated by a piecewise function of two Gaussians:

$$p(x|y) = \begin{cases} \mathcal{L}(x), & \text{if } y < y_{\gamma\%} \\ \mathcal{G}(x), & \text{otherwise} \end{cases} \tag{3.6}$$

where $y_{\gamma\%}$ is equal to a pre-specified percentile of the observed y response values (e.g. $y_{50\%} =$ 50th percentile of the observed responses). TPE uses Gaussian Processes (GP) to model each of $\mathcal{L}$ and $\mathcal{G}$: $\mathcal{L}$ contains all observations with $y < y*$, $\mathcal{G}$ contains all observations with $y \geq y*$.

Gaussian Processes (GP) used for regression. The regression model is an interpolated model from a set of p-dimensional sample points and their target scalar values. It is a desirable approach for high-dimensional, sparse sampling settings because it "lets the data speak" by simultaneously using both local information (where data is densely sampled) and far distant information (where data is sparsely sampled) from the data. It is more non-parametric than most (but not all) regression models.[7]

The GP method is not sparse as all data samples used to build the model are also used during prediction. In the labeling phase, the distance between the incoming sample and each sample in the training set (of size n) must be computed. The labeling phase has a computational complexity of $O(n)$ for each sample to label. When the training set is large, GPs require $O(n^3)$ operations to build the model (due to the computation of an inverse for K), so it is not feasible for large training sets [Williams, 1998]. Also, the method is not efficient in very high dimensional spaces (i.e. $p > 100$). But as each hyperparameter has its own TPE model, independent of the other hyperparameters, the high dimensionality problem is not a concern.

Fully understanding the TPE approach requires understanding of estimation using Gaussian Processes (GP). The explanation of GP here is in the context of regression with one dependent variable: $\boldsymbol{x} \mapsto y$. The training data D contains n tuples of vector inputs and scalar outputs (Equation 3.7). For additional details of Gaussian Process techniques beyond what is provided

[7] An entropy-based method would be more non-parametric. The choice of kernel can impart a bias on the model (e.g. it is possible to model periodic functions with a carefully constructed kernel function that includes trigonometric operators).

here, consult [Rasmussen, 2006].

$$D = \{\langle \boldsymbol{x}_1, y_1\rangle, \ldots, \langle x_n, y_n\rangle\} \qquad X = \begin{bmatrix} \boldsymbol{x}_1 \\ \cdots \\ \boldsymbol{x}_n \end{bmatrix} \qquad Y = \begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix} \tag{3.7}$$

A Gaussian Process is a subtype of stochastic process made of a collection of n random variables: $F(X) = \{f(X_1), \ldots, f(X_n)\}$. Each of these n random variables is a multivariate Gaussian normal distribution in p-dimensional space. The stochastic process of a Gaussian Process can be fully described using only the means and $(n \times n)$ covariance matrix computed from a finite subset of sample points. Each of the sample points is assumed to have been generated from this process. With this information, it is possible to determine the probability distribution on the dependent variable y for a new sample point specifying values for the p dimensions in $\boldsymbol{x}$.

A dataset can be thought of as a set of samples taken from the Gaussian Process. Each sample point is assumed to have been generated from this process, and, specifically, is assumed to be the mean of a Gaussian distribution that corresponds to one of the n random variables. The measured covariance between each pair of sample points is used to construct the covariance matrix using a kernel function. The choice of kernel function ($k(x, x')$) depends on the notion of distance in the space and is domain dependent. But a common choice[8] is a squared exponential (also known as a Radial Basis Function or Gaussian) of the form:

$$k(x, x') = \underbrace{\sigma_f^2 \exp\left[\frac{-(x - x')^2}{2l^2}\right]}_{\text{distance component}} + \underbrace{\sigma_n^2 \delta(x, x')}_{\text{sampling noise component}} \tag{3.8}$$

$$\delta(x, x') = \begin{cases} 1, & \text{if } x = x' \\ 0, & \text{if } x \neq x' \end{cases} \tag{3.9}$$

where σ_f^2 is the maximum allowable covariance, l is the length parameter, and σ_n^2 is measurement noise (usually obtained empirically from the input data). $\delta(x, x')$ is the Kronecker delta function and is used to incorporate uncertainty due to sampling noise. The kernel is a similarity measure between pairs of sample points. Samples very close to each other will have high covariance, and samples very distant will have low covariance. It is also possible to have complex kernels that account for periodicity or special geometric relationships that modify the similarity measure for domain specific reasons.

[8] Standard choices for this are squared exponential ($\exp(-d^2)$), absolute exponential ($\exp(-|d|)$), generalized exponential, cosine ($\cos(d)$), cubic ($(\alpha - d)^3$), and linear ($\alpha - d$) where $d = x - x'$.

GPs are used for regression by interpolating from the set of samples. Given a new candidate sample value $x*$, we would like to estimate the distribution of the target $y*$ and determine the most likely value. We assume this new sample is also sampled from the GP. The training data and the estimate $\bar{y}*$ for a new point $x*$ can be written as the distribution in Equation 3.10:

$$\begin{bmatrix} Y \\ y* \end{bmatrix} \approx \mathcal{N}\left(0, \begin{bmatrix} K & K_*^T \\ K_* & K_{**} \end{bmatrix}\right) \tag{3.10}$$

$$K = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_m, x_1) & k(x_m, x_2) & \cdots & k(x_m, x_n) \end{bmatrix} \tag{3.11}$$

$$K_* = [k(x_*, x_1) \dots k(x_*, x_n)] \qquad K_{**} = k(x_*, x_*) \tag{3.12}$$

$$\bar{y}* = K_* K^{-1} Y \tag{3.13}$$

$$var(y*) = K_{**} - K_* K^{-1} K_*^T \tag{3.14}$$

All diagonal elements in K are $\sigma_f^2 + \sigma_n^2$. Off diagonal pairs with large distances will be near 0. Given this system of equations, the only free variable is $y*$, so the distribution of $y*$ is determined by the system. The most likely value of $y*$ is estimated using the maximum likelihood estimate which is the mean of the distribution (computed in Algorithm 9, lines 4-5). Error bars for the true value of $y*$ can be estimated using $\bar{y}* \pm 1.96\sqrt{var(y*)}$ for the 95% confidence level.

Algorithm 9: EstimateScore Algorithm Listing — `estimateScore`(U, x)

Data: Y training set target values, K Gaussian Process model, $k(\bullet, \bullet)$ kernel function, U model configuration
Result: Estimate $(\bar{y}*)$

```
1 unpack GP model parameters
2 ⟨K, Y, θ⟩ = U
3 compute MAP estimate for given value of x
4 K* ⟵ {k(x*, x_1, θ), ..., k(x*, x_N, θ)}
5 ȳ* ⟵ K* K^{-1} Y
6 return ȳ*
```

The kernel parameters σ_f^2, l, and σ_n^2 are the Gaussian Processes' model parameters and are collectively called θ. In practice, σ_f^2 and l are determined from a calibration process that optimizes the fit of Equation 3.10 on the input data (Algorithm 10, line 3). The maximum a posteriori estimate for θ will maximize $P(\theta|Y, X)$. Also assume that all configurations to be equally likely (no reason to prefer any particular θ over another), so $P(\theta)$ is uniform. Using

Algorithm 10: BuildGaussianProcess Algorithm Listing — `buildGaussianProcess` (Q)

Data: Q training dataset, $k(\bullet, \bullet, \bullet)$ kernel function, θ kernel parameters
Result: Gaussian Process model ($\langle K, Y, \theta \rangle$)

1 $\{\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \ldots, \langle x_N, y_N \rangle\} = Q$
2 $Y = \{y_1, y_2, \ldots, y_N\}$
3 $\underset{\theta}{\text{maximize}} \quad -\frac{1}{2} Y^T K^{-1} Y - \frac{1}{2} \log |K| - \frac{n}{2} \log 2\pi, \quad \text{where } K = \begin{bmatrix} k(x_1, x_1, \theta) & \cdots & k(x_1, x_n, \theta) \\ k(x_2, x_1, \theta) & \cdots & k(x_2, x_n, \theta) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1, \theta) & \cdots & k(x_m, x_n, \theta) \end{bmatrix}$
4 **return** $\langle K, Y, \theta \rangle$

Bayes' rule it is possible to find this by maximizing the quantity $P(Y|X, \theta)$.

$$P(Y|X, \theta) = \frac{P(\theta|Y, X)P(Y, X)}{P(\theta)P(X)} \tag{3.15}$$

The log likelihood can be maximized as

$$\log P(Y|X, \theta) = -\frac{1}{2} Y^T K^{-1} Y - \frac{1}{2} \log |K| - \frac{n}{2} \log 2\pi. \tag{3.16}$$

This can be maximized using a standard multivariate optimization algorithm such as gradient descent. The maximum of the log likelihood for all values of θ determines the parameter configuration to use for predicting $y*$ for unlabeled sample $x*$. GPs assume a model of the form

$$y* = \mathtt{f}(x*) + \mathcal{N}(0, \sigma_n^2) \tag{3.17}$$

where $\mathcal{N}(\mu, \sigma^2)$ is a standard normal distribution with mean μ and variance σ^2, and $\mathtt{f}(\bullet)$ is the underlying generating function. The model assumes all samples have measurement noise with variance σ_n^2. This optimization determines the contribution of each sample from the dataset when predicting for a new sample. This optimization is only done once (when the model is generated). The computed model θ is then used for all predictions.

Geometric interpretation of Gaussian Processes. One interpretation of Gaussian Processes is that the model is a collection of p-dimensional Gaussians with one centered at each of the n sample points. A stylized diagram of Gaussian Process regression is shown in Figure 3.4. The optimization process to determine θ sets the spread of the Gaussians (and any other model parameters). The spread used is equal for all n distributions. This optimization process maximizes the "fit" of the model to the dataset. The predicted $\bar{y}*$ value is a linear combination of the y values for the n training points. The contributions are computed in Equation 3.13. Samples that are more similar to $x*$ will have a greater effect on the prediction of $y*$ as the covariance between the samples (i.e. $k(x*, x_i)$ for $i \in \{1, \ldots, n\}$) is higher. The Gaussian Process method

is called a linear smoother for this reason [Hastie and Tibshirani, 1990].

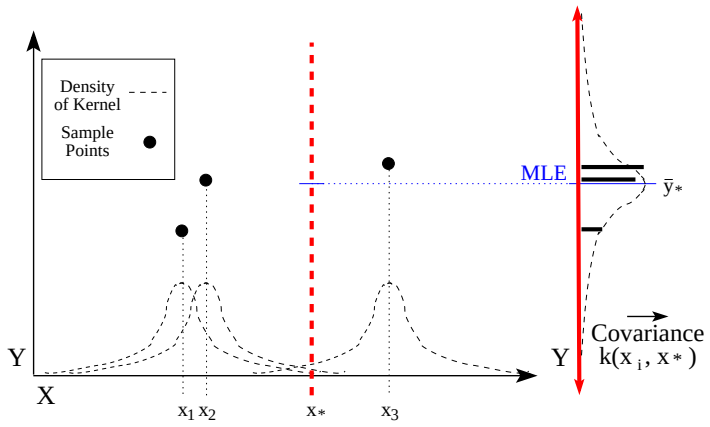


Figure 3.4: A stylized Gaussian Process example showing the Maximum Likelihood Estimate for $y*$ given three data points. The probability density function for the target variable y given a $x*$ is shown at the right. The density function curves shown are for a Radial Basis Function kernel in one-dimensional data.

Oversearching. Non-exhaustive search schemes are applied to feature selection to ensure that a search can be performed even for very large search spaces. The non-exhaustive search schemes can also have the additional benefit of reducing oversearching. In model-based machine learning, searching a very large space of possible configurations can lead to poor generalization performance of the model. *Oversearching* in model optimization is the phenomenon of a search process exploiting transient patterns in the dataset. By trying many possible model configurations, the search increases training set accuracy but does not generalize to unseen samples. Quinlan and Cameron-Jones [1995] note that "expanding search leads eventually to a decline in predictive accuracy as idiosyncrasies of the training set are uncovered and exploited." In the experiments, the algorithm's search extent is varied by setting the beam width in a beam search (a memory-limited variant of best-first search). The oversearching problem is orthogonal to the problem of overtraining (increasing model precision and complexity to the extent that generalization performance on new samples is degraded) as oversearching can occur even for simple models.

3.4 DGFS Discussion

Developer-Guided Feature Selection is most relevant for domains where the number of features is large and the number of observations is limited. The relationships between the feature selection method, the number of training set observations, and the number of features is shown in

Figure 3.5. It is not always the case that DGFS is beneficial to the model construction. This figure illustrates the broad conditions under which DGFS is most useful.

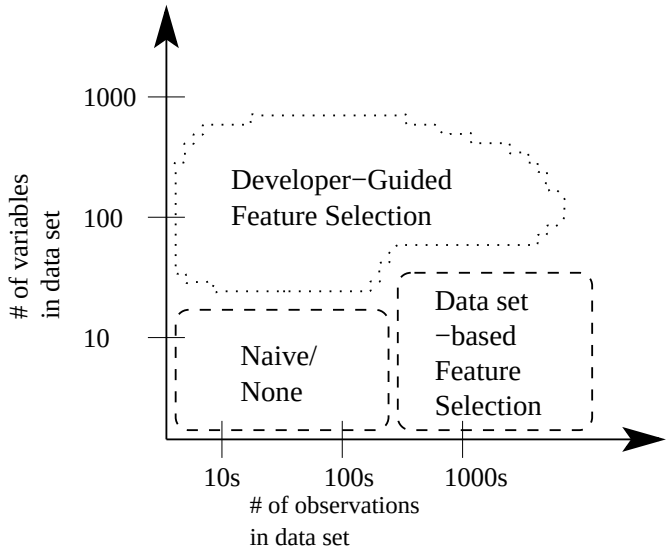


Figure 3.5: Region of greatest effectiveness for feature selection methods relative to the feature set size and training set size.

We seek to address challenges in real-world application domains where a prediction or generalization is desired. The essential contributions of DGFS are the following:

1. The inclusion of time-delayed observations (that we call *lagged features*) as elements in the feature vector, in addition to the most recent observed values for the features. This enables us to capture temporal dependencies among the features, but this innovation alone tends to produce a model with large numbers of features and a model that is not accurate.

2. The construction of a hierarchical structure among the features incorporates domain knowledge. It also facilitates pruning of the features that do not improve predictions. To accomplish this, we enumerate systematically the different subsets of features and examine their performance, selecting combinations that produce higher quality predictions. We have demonstrated that this feature set pruning works in domains where the features have a temporal aspect. This is also successful for domains which contain significant spatial relationships between features.

3. By examining the performance of candidate lag schemes and the corresponding performance of each, it is possible to extract domain knowledge. Specifically, it is possible to determine the relevance or irrelevance of individual features by observing their inclusion

in the lag scheme. Also, in domains where temporally lagged instances are included in the feature set, it is possible to determine temporal relationships between features.

4. While the previous contributions are—in principle—not tied to the choice of underlying machine learning algorithm, the following does rely on features of the base algorithm used in our experiments: Partial Least Squares regression (PLS). By examining the weights computed by the PLS model learning phase, it is possible to determine the relative significance of the chosen features.

The DGFS approach has desirable properties for application domains involving real data. In the next four chapters, the experimental performance of our framework is analyzed in the context of four complex domains.

3.5 Overview of Experiments

Table 3.1: Feature selection processes ("FS") undertaken for each data domain presented in this document. ✓ denotes this FS-domain pair is evaluated in the experiments. ✗ denotes no evaluation of the pair.

Domain \ FS	No FS	Data Driven		DGFS			
	All Lag	CFS	BFS	Exhaustive	Greedy	Random	Guided
1. Synthetic (Ch. 4)	✓	✓	✓	✓	✓	✓	✓
2. Airline (Ch. 5)	✓	✓	✓	✓	✗	✗	✗
3. River Flow (Ch. 6)	✓	✓	✓	(some)	✓	✓	✓
4. Supply Chain (Ch. 7)	✗	✗	✗	✓	✗	✗	✗

In these experiments, we find that different domains necessitate different approaches. For simpler domains having fewer variables and time lags, an exhaustive feature selection search is tractable, so only that method was employed. For larger domains, exhaustive search is intractable (even with feature class constraints) given the size of the configuration space. In these cases, greedy methods have been employed both to increase efficiency and to make DGFS tractable. These methodological choices are shown for all domains in the Table 3.1.

Chapter 4

Applied Domain: Synthetic Data

While empirical algorithm evaluation using real datasets is critical in determining real-world effectiveness, empirical evaluation alone does not readily explain the reasons for good or poor model performance on a particular dataset. Ideally, a well performing model should be similar in composition to the correct model for the target concept. The difference between a calibrated model[1] and the underlying "true" model is often unknowable because the latent generating function from which the data was generated is not known. In complex datasets, a prediction model could perform well because it accurately models the target concept. Alternatively, it could be leveraging idiosyncrasies of the dataset to make more accurate predictions, or some combination of both.

In an effort to provide objective evidence that the feature sets obtained by the feature selection search process are related to the correct (i.e. ground truth) feature set, this chapter builds a controlled model recovery simulation on synthetic data. In this chapter, we perform model calibration and feature selection on artificial datasets for which the generating function is known exactly. This facilitates measurement of the correspondence between the best observed model and the ground truth. This is used to justify the design of the UGFS framework and explain the rationale behind several design choices. These artificial datasets mimic the properties of the natural applied domains used in subsequent chapters (e.g. river flow, supply chain, etc.). These synthetic data sets are also used to explore various dataset splitting techniques found in the literature.

The following terminology is used for describing these synthetic datasets. An *independent variable* is a variable whose value is not determined by any other variable observed in the domain. A *dependent variable* is a variable whose value depends on a polynomial combination of zero

[1]The calibrated model refers to all determined aspects of the model including the modeling function choice, feature set, and model parameters.

or more variables. The target variable is a dependent variable. The two types are mutually exclusive and are complete: each variable is either an independent or a dependent variable.

There are two synthetic data domains used for experimentation in this chapter (Table 4.1): Multi-Concept and Synthetic River. The former artificial domain examines two principal aspects: 1) the effect of the model testing configuration on measured performance, and 2) the side-effects of irrelevant variables on the calibrated model. The latter domain is a system populated with 8 independent random walks that are combined in a tree-like structure with no cycles. This system is intended to mimic the physical relationships in a river network. There are 8 independent random walk variables, of which 7 are combined (with various time lags) to compute the target variable.

Table 4.1: Properties of the artificial data domains introduced in this chapter.

Data Set Name	# of Relevant Vars	# of Vars	Has Lagged Variables
Multi-Concept	5	10	No
Synthetic River	14	15	Yes

4.1 Model Testing Configuration Approaches

There are many rules-of-thumb in the literature for splitting time-dependent datasets for the purposes of modeling and prediction. The most appropriate splitting method is, to some extent, a decision driven by the model developer's knowledge and understanding of the application domain. The risk of choosing an inappropriate method is that the prediction performance estimates found in the testing phase will not be representative of performance on new never-before-seen observations. The measured generalization performance in the testing phase could be better or worse than the true model performance on novel data. This phenomenon can be affected by both the splitting criteria and the machine learning algorithm. In situations where the computed model will later be used for on-line predictions, this aspect of uncertainty is important to consider.

Broadly, the these risks are caused by:

1. information leakage from the test set to the training set,
2. the utilization of concepts that could not yet have been discovered due to temporal dependency in the dataset, or

3. the model overfitting the data due to the presence of more (or fewer) concepts than actually could have been known at a particular time point.

Time series datasets are ubiquitous in nature. When modeling concepts with temporal data, machine learning practitioners should pay special attention to their properties. First, these data are not *i.i.d.* as adjacent samples will have statistical covariances with each other. These temporal dependencies can be leveraged through, for example, the construction of time-shifted variables ("lagged variables") in the feature set which is fed to the learning algorithm. Second, while the statistical dependencies should be considered in prediction, using information that is only available forward in time of the current observation is inappropriate in some scenarios. For example, in river flow prediction, building a prediction model that uses observations ahead of the current time, would make the model less useful for on-line prediction tasks as some required inputs would be unavailable in the online setting. In other sequential domains, this use of later information may be appropriate. For example, in text processing where decisions are made at the per word level, using information about the next word may be perfectly appropriate.

This issue of avoiding future knowledge is also important in the composition of the training set used for calibration. For temporal domains with rapidly changing concepts, it is unlikely that a deployed machine learning model will have any knowledge of future (not yet observed) concepts. A model trained on observations from across the entire sampling period (e.g. trained using cross validation for data splitting) could have a higher prediction accuracy than a model trained on only samples in the period preceding the current sample. It is also possible for the model to have lower than expected accuracy when it is trained on multiple concepts. The concepts may conflict with each other in the calibration process resulting in lower prediction accuracy. When samples are *i.i.d.*, this is not a concern, but in domains with temporal dependence the performance results are affected.

In practice, many different data splitting methods are in common usage. Figure 4.1 provides stylized diagrams of the methods. We enumerate and describe each in the text below:

- (a) all-in-sample — A simple approach to prediction model evaluation is to predict the target value of each training sample using the model after the model is calibrated. This corresponds to a model testing configuration with training, validation, and testing all as the same set. The advantage of this approach is that all the data is used in calibration and evaluation so interesting samples will be both incorporated into the model and will be part of the final score. The principal risk of this approach is overfitting because the calibration and hyperparameter optimization will optimize for lowest evaluation error. The test set score will not be representative of scores for yet-to-be-observed samples as the model may be optimized for the idiosyncrasies (outliers, range of values, etc.) in the training set. In the experimental design in which there are separate training, validation,

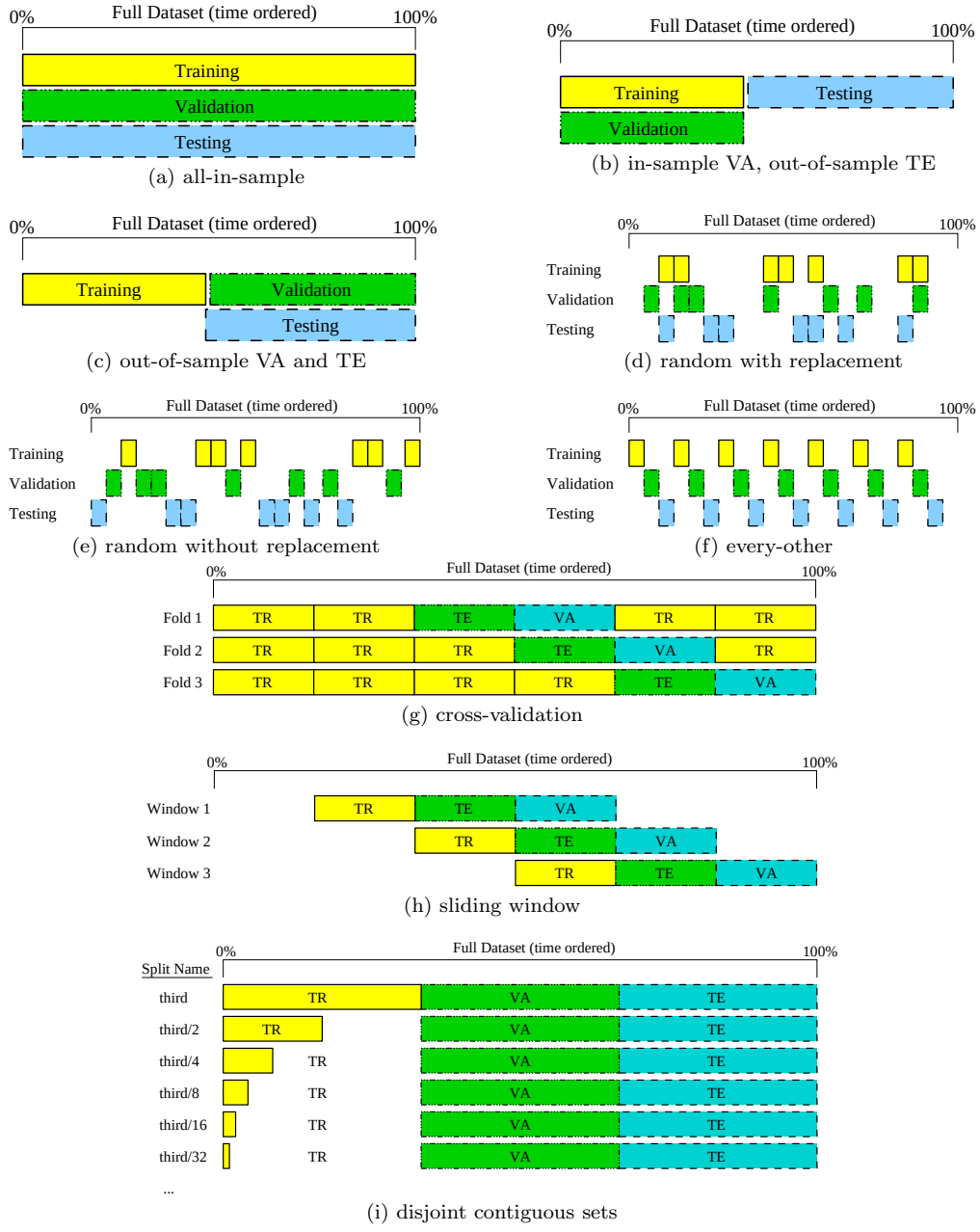


Figure 4.1: Data splitting methods. Note: TR = Training, VA = Validation, and TE = Testing.

and testing phases, the process is a follows. First, the model is calibrated using all samples in the dataset. The model hyperparameters may be modified based on validation set stores computed over the entire data set (in-sample scoring). Finally, the optimized model's score is based on prediction errors computed over the entire dataset.

- (b) in-sample validation, out-of-sample testing — In this regime, the calibration and hyperparameter optimization is done over the same set. The final scores are created on a "hold out" test set. This approach also risks overfitting on the hyperparameter optimization, but the final reported scores should be representative of scores on unseen samples.

- (c) out-of-sample validation and testing — This regime employs the same set for both validation and testing. In our formulation of this regime, the training set is both earlier in the data sequence and is disjoint from the validation set. The model is less likely to overfit but the hyperparameter optimization may overfit by leveraging idiosyncrasies in the validation set to obtain a better final score. Again, this may result in better than should-be-expected final scores.

- (d) random with replacement — This method randomly samples three sets (training, validation, and testing) from the original dataset. Performance is similar to all-in-sample and every-other but will have additional randomness in the evaluation scores due to the random sampling behavior.

- (e) random without replacement — This method randomly assigns each sample to one of training, validation, or testing. The resulting sets are disjoint. But for non-*i.i.d.* data, this will be similar to all-in-sample.

- (f) every-other — In the two set context (training and testing only), this is called even-odd. The first sample and every three observations afterward in the sequence are assigned to the training set. The two other sets are assigned similarly. This approach is very likely to leak information in time-series and multi-concept contexts. But it is advantageous in having less randomness induced in the result as compared to random with replacement (d) and random without replacement (e).

- (g) cross-validation — Cross-validation is an ubiquitous approach for a model testing configuration in *i.i.d.* domains from the machine learning literature. It efficiently maximizes the amount of data used for calibration while simultaneously scoring over all samples in the dataset. In sequential or time-series data, this method risks using future concepts on earlier observations. It leverages persistent properties of data over long sequences. Unfortunately in multi-concept sequential data, using many concepts to train the model can

result in a poor model fit. Another disadvantage is that is requires many calibrations (equal to the number of folds, i.e. n-fold cross-validation) to complete the evaluation. The most extreme case (when the number of folds is equal to the number of observations) is called leave-one-out cross-validation.

- (h) sliding window — Sliding window evaluation uses a sequential mask ("window") over a portion of the dataset to determine a training, validation and test subset. The window is shifted several times across the dataset to obtain several evaluations across the entire data sequence. This method is very useful for time series and sequential data because it does not leak future concepts, keeps the training set small to avoid poor calibration on many concepts simultaneously, and minimizes the distance between samples for training and testing. It can require many calibrations to score the entire dataset (similar to cross-validation). If the dataset is small (i.e. few observations in the training set) this model risks poor fit. Because of the advantages on time-series data, this methods is used extensively in this thesis for natural data domains.

- (i) disjoint training, validation, and testing — Using separate, contiguous sequences for training, validation, and testing is one approach to avoid overfitting. This approach is very common in cases of limitless data (such as in synthetic domains). A disadvantage is that it is not efficient in terms of the amount of data as only one-third of the dataset is used for training which may be problematic when the dataset is small. For large datasets with unchanging concepts, this is a suitable approach.

There are many tradeoffs to consider when choosing a model testing configuration, and there is no general method that is best for all situations. The disadvantages are attributed to the splitting methods in Table 4.2. In non-*i.i.d.* settings where concepts are changing over time, it is better to use a method with a smaller training set to capture a single concept well (with little noise). Also, it is important to avoid leaking information from later observations by using it on earlier samples. The essential conflict between large training set sizes and noise introduced by a large multi-concept training set is the primary conflict in this section.

	Risks					
Method	**leaks test set into training**	**leaks future concept into training**	**poor fit on multi-concept**	**additional randomness added**	**large number of calibrations to evaluate**	**evaluation score considers subset of data**
(a) all-in-sample	•	•	•			
(b) in-sample VA out-of-sample TE	•		•			•
(c) out-of-sample VA and TE			•			•
(d) random with replacement	•	•	•	•		•
(e) random without replacement		•	•	•		•
(f) every other		•	•			•
(g) cross validation		•			•	
(h) sliding window					•	
(i) equal contiguous disjoint			•			•

Table 4.2: Splitting method risks. Note: VA = Validation and TE = Testing.

4.2 Multi-Concept

In real-world data, situations where the concept to be learned is changing over the course of the dataset's time range are common. The objectives of this scenario are to observe the effects of common splitting criteria on a multi-concept dataset and also to observe the effects of irrelevant variables used in the model. This scenario contains up to fifteen additional independent variables. These additional variables are independent random walk variables which are not in the target variable computation. For some combinations of machine learning model and model testing configuration, the additional independent variables will provide predictive power.

This section explores the topic of model testing which is often not addressed when applying machine learning methods to sequential data. Popular supervised machine learning algorithms commonly assume that the observed samples used in training are independent and identically distributed (*i.i.d.*) [Dietterich, 2002]. In an *i.i.d.* setting, the value of the previous observation does not provide any information on the value of the current observation, and the set of observations is a representative sample of the space. In practice, these assumptions are rarely true in real-world machine learning scenarios but the methods are nevertheless applied anyway. This exposition examines a synthetic multivariate time series domain that is known to have four distinct concepts (relationships between the independent variables and the dependent variable) over the course of the time series. This artificial data domain allows us to compare common approaches for splitting datasets for machine learning model calibration and testing in the presence of concept drift. Changing concepts are challenging to learn in some natural data domains. We explore responses to this problem.

4.2.1 Multi-Concept Generating Function

The purpose of the Multi-Concept domain is 1) to examine how different splitting methods perform relative to each other and 2) to examine the effect of additional (possibly) correlated but independent variables on prediction performance. To ensure the generality of our conclusions, 10 separate datasets ("seeds") conforming to this specification are constructed. Each seed has a unique set of randomly generated initial conditions. Each seed has a sequence length of 10,000 time units and the initial value of each random walk variables (named "rwX" and "irXX") is drawn uniformly from the variable's range.

The generating function for the dependent "target" variable ($dv1$) is given in equation 4.1:

$$dv1 = rw1 + concept1on * rw2 + concept2on * rw3 + concept3on * rw4 + concept4on * rw5 + \mathcal{N}(0, \sigma_\gamma) \tag{4.1}$$

where $\mathcal{N}(\mu, \sigma)$ is the Gaussian normal distribution (with mean μ and variance σ^2) and the

remaining variables are defined in Table 4.3.

Variable Name	Description	In Feature Set
rw1,...,*rw5* =	RandomWalk(min=10, max=200, stepsize=2, drift=8)	•
ir01,...,*ir15* =	RandomWalk(min=10, max=200, stepsize=2, drift=8)	•
concept1on =	$\begin{cases} 1 & 0 \leq t \leq 4000 \\ 0 & \text{otherwise} \end{cases}$	
concept2on =	$\begin{cases} 1 & 500 \leq t \leq 5500 \\ 0 & \text{otherwise} \end{cases}$	
concept3on =	$\begin{cases} 1 & 5000 \leq t \leq 10000 \\ 0 & \text{otherwise} \end{cases}$	
concept4on =	$\begin{cases} 1 & 5500 \leq t \leq 10000 \\ 0 & \text{otherwise} \end{cases}$	

Table 4.3: Underlying generation function for Multi-Concept

There are also variants of the scenario labeled from 00 to 15 with the number indicating the number of additional independent variables (named as irXX, where XX $\in \{01, \ldots, 15\}$) that are also included in the input. The values for each of the "random walk" variables (named "rwX" or "irXX") is generated from a random walk process. Equation 4.2 describes the random walk model with stochastic drift used in data generation.

$$y_t = \underbrace{y_0}_{\text{initial value}} + \underbrace{\sum_{i=-\text{drift}}^{0} f(t+i)}_{\text{stochastic trend}} \tag{4.2}$$

$$f(i) = X \tag{4.3}$$

$$P(X) = \begin{cases} 0.5 & \text{, if } X = \text{step} \\ 0.5 & \text{, if } X = -\text{step} \end{cases} \tag{4.4}$$

The specifications of the variables for these seeds including the time ranges of each concept are given in Table 4.3. A time series plot of one of the seeds is given in Figure 4.2. This provides a look into the behavior of the data over time.

4.2.2 Multi-Concept Experiments

For these experiments, we build various models to predict the value of $dv1$ given all values for the independent variables in the domain at each time step. (There are no time offsets in the generating function, and no time lags are considered for this scenario.) The learning algorithm is Partial Least Square (PLS) Regression, a linear regression model, but the results will be similar for other linear regression methods. The underlying concepts are linear as well, so this choice is

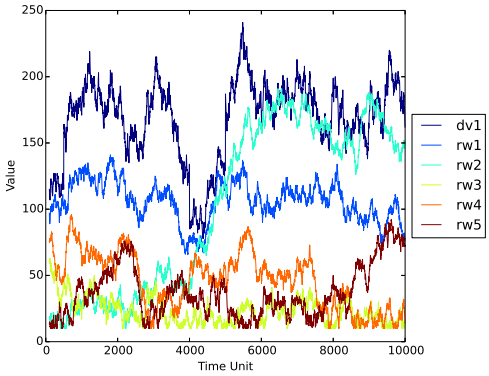


Figure 4.2: Time Series for Multi-Concept

suitable. Table 4.4 shows the mean statistic and standard deviation of the prediction errors over all samples in all 10 seeds. The prediction score is computed as the mean error of all predictions made:

$$\text{score} = \frac{1}{n}\sum_{i=1}^{n}\frac{|\text{est}_i - \text{act}_i|}{|\text{act}_i|}, \text{ for relative scoring.} \tag{4.5}$$

The score is computed relative to the true value for each sample.[2] These results show that the measured performance can vary greatly even for the same data and learning algorithm.

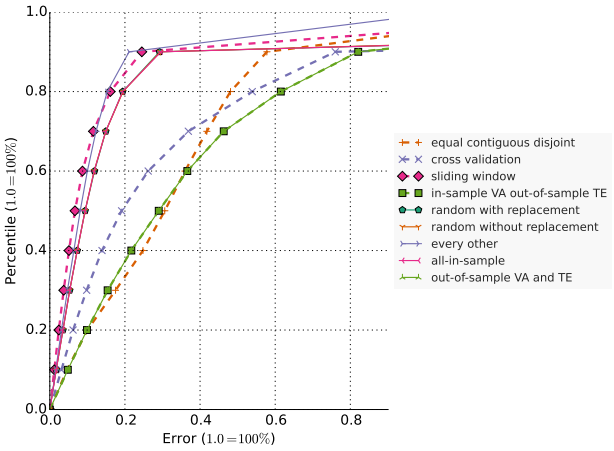
Table 4.4: Performance comparison of common data splitting approaches in the Multi-Concept dataset using PLS Regression. Note: values reported are relative error (where 0.1 = 10%).

Method	**Mean Error (relative)**	**Std. Dev.**
equal contiguous disjoint	0.315	0.212
cross validation	0.327	0.410
sliding window	0.109	0.140
in-sample VA out-of-sample TE	0.373	0.321
random with replacement	0.152	0.250
random without replacement	0.153	0.250
every other	0.114	0.130
all-in-sample	0.153	0.252
out-of-sample VA and TE	0.373	0.321

The aggregate statistics only show a limited perspective of these results. The distributions of the errors experienced by each splitting method differ as well and are shown in Figure 4.3.

[2]This is consistent with measurements of physical systems such as stream flow where the measurement error is proportional to the true value [Singh, 2013].

Figure 4.3: Error cumulative distribution function (CDF) for various data splitting methods.



Broadly the splitting methods cluster into two distinct classes:

1. all-in-sample, random with replacement, random without replacement, every other, sliding window, or

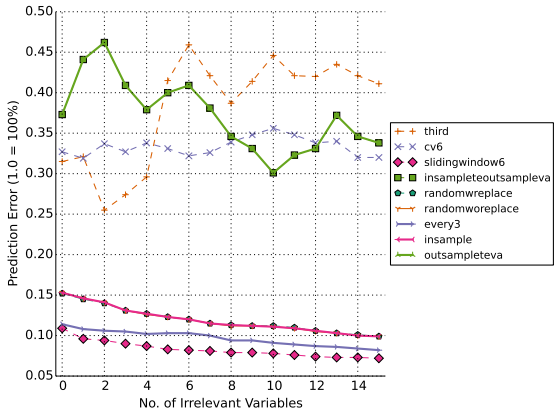
2. equal contiguous disjoint, cross validation, out-of sample VA and TE, in-sample VA out-of-sample TE.

The methods of the all-in-sample cluster benefit from having observed all (or most) of the dataset previously and are able to leverage idiosyncrasies[3] of the generating function. The all-in-sample cluster can learn from all concepts simultaneously, so it can use knowledge of concepts that appear after the current test set period. This knowledge about the future would not be available in an on-line setting. This aspect of the model can be observed by looking at the mean prediction error observed at each sequence number across all 10 trials. The sliding window approach in this experiment employs 4 window ranges in total, the edges of the windows are denoted by vertical lines. The errors near the end of each window tend to be higher than the errors near the beginning of the window, this is because the concept may change over the range of the window, so the model's concept will be less applicable as the distance between the current observation and the model's training set increases.

Irrelevant variables in the input set also can effect the splitting methods differently. Figure 4.4 shows how the mean error statistic varies as additional independent variables are added to the input. The figure shows the effects of including irrelevant variables in the augmented dataset

[3]In this data, it happens that at either zero or one concept is present at each time step. This can be leveraged by the learning model if given training data over all concepts.

Figure 4.4: Mean errors of common data splitting methods in the presence of a varying number of irrelevant variables using PLS Regression as the underlying learner.



provided to the learner for various splitting approaches. The sliding window cluster methods all benefit from the addition of irrelevant variables. We conjecture that this is because the irrelevant variables allow the learner to track the ordering of input observations and allow for memorization of some values. In conclusion, irrelevant variables can effect the concepts learned but performance is also dependent on the splitting approach utilized. In time series domains with changing concepts, we find that sliding window approaches are generally preferred as it provides the most accurate learner that is less likely to be contaminated by possibly irrelevant concepts.

Multi-Concept Conclusions

This section provided a study of various approaches of splitting data sets for machine learning that possess temporal dependencies. The risks of leaking information across training and testing is evaluated for each splitting method. The ideal case for a changing concept data set is an approach that does not leak data from future concepts. The best possible approach that does not leak data is a sliding window regime that is retrained at each time step, the window size of the training set is varied experimentally to find the best size. However, retraining at each time step is a very expensive approach, so retraining periodically is the alternative taken in these experiments.

In applications where this kind of uncertainty is present, the solution may be to perform experiments similar to the ones shown to determine the sensitivity of the model to the splitting criteria for the application domain.

4.3 Simulated Spatio-temporal Domain: SyntheticRiver

This experiment shows the effect of having a limited quantity of training data. Clearly, less information will at some level effect the prediction model's accuracy. But there is also an interaction between feature selection approach chosen and the model's performance as training data is reduced. The SyntheticRiver scenario contains both independent and dependent variables (specifically, summations with time delays of other independent and dependent variables). The system mimics the behavior observed in real physical systems. This scenario is most similar in the relationships and time offsets to those found in the real-world application domain of river/stream flows prediction described in Chapter 6. In the generating function for this dataset, we explicitly encode the time delays, variable dependencies, and observation noise to facilitate benchmarking of feature selection methods.

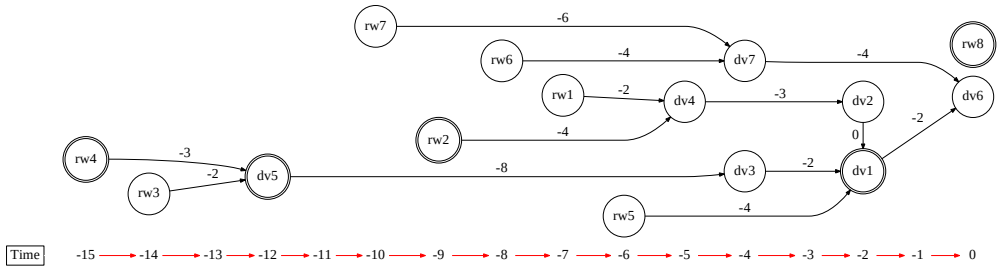


Figure 4.5: Generating function diagram for SyntheticRiver network. Double circle nodes denote observable variables.

For these experiments, the regression model attempts to predict the flow at the *dv1* location for 7 time steps ahead (time $t+7$) of the current observations (time t). The independent variables are random walk processes with a dynamic trend. A visual diagram of the relationships and offsets is shown in Figure 4.5. Only a subset of the variables is available for prediction (as is true in many real physical systems), and these nodes are denoted by double circles.

The mathematical relationships for this system are given in Table 4.5. All observable values have 1% Gaussian noise applied ($\mathcal{N}(\mu = 1.0, \sigma = 0.01)$) to simulate measurement error that is proportional to the magnitude of each measurement.

As this experiment is concerned with the effect of limited training data, the schedule for limiting the training set size is given in Table 4.6. In these experiments the length of the maximum length training, validation, and test sets is 3300 time steps in each. For the reduction experiments, the training set is truncated based on the schedule. The validation and test sets remain the full length and contain the same samples as the full length version to facilitate paired-sample statistical significance testing. This ensures comparisons between methods are reasonable as the comparison results are for precisely the same set of samples. Unlike the experiments of the

Variable Name	Description
$rw1 =$	RandomWalk(min=10, max=20, stepsize=1, drift=10)
$rw2 =$	RandomWalk(min=10, max=200, stepsize=1, drift=10)
$rw3 =$	RandomWalk(min=10, max=100, stepsize=1, drift=10)
$rw4 =$	RandomWalk(min=10, max=60, stepsize=1, drift=10)
$rw5 =$	RandomWalk(min=10, max=200, stepsize=1, drift=10)
$rw6 =$	RandomWalk(min=10, max=200, stepsize=1, drift=10)
$rw7 =$	RandomWalk(min=10, max=200, stepsize=1, drift=10)
$rw8 =$	RandomWalk(min=10, max=200, stepsize=1, drift=10)
$dv1 =$	$\phi_0(dv2) + \phi_{-2}(dv3) + \phi_{-4}(rw5)$
$dv2 =$	$\phi_{-3}(dv4)$
$dv3 =$	$\phi_{-8}(dv5)$
$dv4 =$	$\phi_{-2}(rw1) + \phi_{-4}(rw2)$
$dv5 =$	$\phi_{-3}(rw4) + \phi_{-2}(rw3)$
$dv6 =$	$\phi_{-4}(dv7) + \phi_{-2}(dv1)$
$dv7 =$	$\phi_{-4}(rw6) + \phi_{-6}(rw7)$

Table 4.5: Underlying generating function for SyntheticRiver. Note: $\phi_c(x)$ is the time delay operator with a time shift of c on variable x.

Table 4.6: Training set size levels for SyntheticRiver experiments

Training Set Size	No. of Observations
1/1	3300
1/2	1651
1/4	825
1/8	412
1/16	206
1/32	103
1/64	51
1/128	25
1/256	13
1/512	6

Table 4.7: Domain-congruent feature class constraints for SyntheticRiver domain.

Variable 1	Constraint	Variable 2
rw2	BEFORE	dv1
dv5	BEFORE	dv1
rw4	BEFORE	dv5
rw5	BEFORE	dv1
rw4	BEFORE	dv1

previous section, we do not examine alternative splitting methods for SyntheticRiver because the underlying concept is known *a priori* to not be changing.

Experiments in this section examine the performance of the proposed (DGFS) and existing state-of-the-art feature selection algorithms presented in Chapter 3 in the context of reduced training set size. Figure A.5 and Table 4.8 show the efficiency curves for each method as the training set is reduced. The scores are computed over prediction error values from 10 randomly generated scenarios (each with different initial random seeds) for 33,000 test set observations in total. In the maximal training set case ("1/1"), each method achieves its best performance. This is expected as this case provides the learning algorithm with ample data to optimize the calibration of the concept even in the presence of noise. The benchmark (no feature selection) approach is labeled "No FS". It is a PLS Regression model with all observable variables included

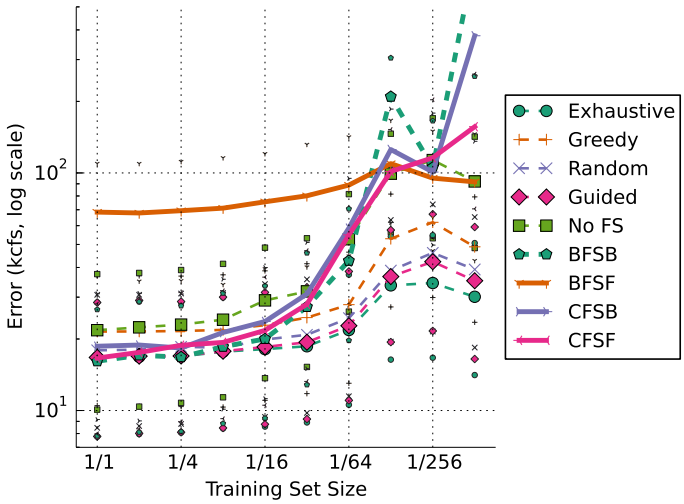


Figure 4.6: Efficiency curve for various methods on SyntheticRiver dataset: error percentiles (25th, 50th, 75th) versus training set size for various feature selection approaches. Errors are presented with a logarithmic y-scale to make differences more visible across the range of error measurements. Small markers denote the 25th and 75th percentiles while the large markers denote the 50th percentile.

and with each variable having all time offsets in the input set (lags $= \{0, -2, -4, -6, -8, -10\}$).[4] The best of the data driven approaches (BFS backward, and CFS forward) as well as all DGFS approaches (Exhaustive, Greedy, Random, and Guided) are able to improve upon the baseline's performance.

As the training set size is reduced, all feature selection methods experience increased errors. Many methods lose accuracy by the "1/16" size trial. The DGFS methods (except for Greedy) are still exceeding the performance of the baseline (No FS at "1/1") from the "1/1" to the "1/64" trials. The DGFS methods' accuracy is ordered roughly in terms of their respective search effort: exhaustive has the least error (and most search effort) and greedy has the greatest error (and least search effort).

Critically, we also find that the performance differences are statistically significant for training sets smaller than the "1/16" level. These statistical accuracy comparisons employ the Wilcoxon Signed-rank Test with Holm-Bonferroni correction to account for the multiple comparisons problem (Table 4.9). Because the test set is large (33,000 observations), the statistical significance tests are likely to find significance when a favorable performance difference exists for the aggregate statistics. For greatly reduced training set sizes, we find that the Exhaustive

[4]For the No FS benchmark, there are 36 input features (6 variables $\times$ 6 time lags).

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Exhaustive	24.7	25.1	24.9	26.0	26.9	27.7	31.8	45.4	45.4	43.2
Greedy	32.2	32.2	32.2	32.4	33.7	35.7	39.8	67.2	79.9	67.5
Random	26.8	27.0	27.1	27.1	29.9	30.6	35.1	52.7	61.1	55.6
Guided	24.8	25.3	25.1	26.0	27.5	28.6	32.4	47.9	56.6	50.5
No FS	32.3	33.1	34.1	35.9	41.5	45.1	67.9	121.1	137.5	115.0
CFSB	26.4	26.7	26.6	30.5	34.4	42.7	76.9	149.6	142.0	494.5
CFSF	24.1	25.5	27.0	27.2	31.4	39.8	72.4	124.8	162.2	210.9
BFSB	22.5	24.4	23.6	26.6	28.3	38.4	57.9	246.4	134.8	1006.7
BFSF	91.5	91.7	92.9	96.1	100.6	108.1	116.0	134.9	122.4	119.2

Table 4.8: Test set root-mean-square error for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain. Values are in kcfs (1000s of cubic feet per second) to correspond to the river domain.

and Guided feature set selection approaches dominate others.

DGFS without constraints

This experiment examines the results of the feature selection search when no feature class constraints are used. The results in Table 4.10 show the experimental results when no constraints are used in the search. The exhaustive search version is not reported because the search space when no constraints are used expands considerably. For larger domains (with many feature classes and possible time lags), exhaustive search will be infeasible without the provided constraints.

Incorrect constraints

DGFS does not require complete correctness of the constraints used in feature selection for satisfactory results. This experiment utilizes a constraint set that is incongruent with the feature relationships in the domain. The constraints used are shown in Table 4.11 and can be compared with the correct constraints in Figure 4.5. The efficiency curves for the DGFS methods using this constraint set are given in Table 4.12. The errors are higher in this experiment when compared to the correct constraint set experiment in Figure A.5, but this highlights the benefit of providing domain-correct constraints on performance results using DGFS. This aspect of incorrect feature class constraints is also examined in the airline ticket price prediction domain in Section 5.5.5.

Expanded Time Lag Choices

Another decision to be made in the DGFS framework is the level of granularity in the time lag choices for each feature class. Up to this point in this section, the lag choices have been of the set $\{0, -2, -4, \ldots, -10\}$ in time units. For this experiment, the granularity has been increased to

Table 4.9: Statistical significance of reduced training set experiments using PLS for the underlying learner. Each symbol in a cell indicates that the row method is more accurate than the symbol's method and that the mean error difference is significant by the Wilcoxon Signed-rank Test with Holm-Bonferroni correction ($\alpha = 5\%$). More symbols in a cell indicate the row method dominates the performance of many other feature selection methods under comparison.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Exhaustive (+)	△ ▽ ◁ ▷ ⋆ □	△ ▽ ◁ ▷ ⋆ □ +	△ ▽ ◁ ▷ ⋆ □ +	△ ▽ ◁ ▷ × ⋆ □ +	△ ▽ ◁ ▷ × ⋆ □ +	△ ▽ ◁ ▷ × ⋆ □ +	△ ▽ ◁ ▷ × ⋆ □ +	△ ▽ ◁ ▷ × ⋆ □ +	△ ▽ ◁ ▷ × ⋆ □ +	△ ▽ ◁ ▷ × ⋆ □ +
Greedy (△)	⋆	▷ ⋆	▷ ⋆	▷ ⋆	▷ ⋆ □	▷ × ⋆ □ +	▷ × ⋆ □ +	▷ × ⋆ □ +	▷ × ⋆ □ +	▷ × ⋆ □ +
Random (▽)	△ ▷ ⋆ □	△ ▷ ⋆	△ ▷ ⋆	△ ▷ ⋆ □	△ ▷ ⋆ □ +	△ ▷ × ⋆ □ +	△ ▷ × ⋆ □ +	△ ▷ × ⋆ □ +	△ ▷ × ⋆ □ +	△ ▷ × ⋆ □ +
Guided (◁)	△ ▽ ▷ ⋆ □	△ ▽ ▷ ⋆ □	△ ▽ ▷ ⋆ □ +	△ ▽ ▷ × ⋆ □ +	△ ▽ ▷ × ⋆ □ +	△ ▽ ▷ × ⋆ □ +	△ ▽ ▷ × ⋆ □ +	△ ▽ ▷ × ⋆ □ +	△ ▽ ▷ × ⋆ □ +	△ ▽ ▷ × ⋆ □ +
No FS (▷)	⋆	⋆	⋆	⋆	⋆	⋆	⋆ □ +	× ⋆ □ □ +		× □ +
CFSB (□)	△ ▷ ⋆	△ ▷ ⋆	△ ▷ ⋆ +	△ ▷ ⋆	▷ ⋆	▷ ⋆	⋆	×	+	×
CFSF (+)	+ △ ▽ ◁ ▷ ⋆ □	△ ▽ ▷ ⋆ □	△ ▷ ⋆	△ ▷ ⋆ □	△ ▷ ⋆ □	▷ ⋆ □	⋆ □	× ⋆ □		× □
BFSB (×)	+ △ ▽ ◁ ▷ ⋆ □ +	+ △ ▽ ◁ ▷ ⋆ □ +	+ △ ▽ ◁ ▷ ⋆ □ +	△ ▽ ▷ ⋆ □ +	△ ▽ ▷ ⋆ □ +	▷ ⋆ □ +	▷ ⋆ □ +		▷ □ +	
BFSF (⋆)								× □	▷ × □ +	× □ +

every integer offset: $\{0, -1, -2, \ldots, -9, -10\}$. The accuracy results for the increased granularity experiment is shown in Table 4.13. This increased granularity provides reduced the errors for the larger training set sizes ("1/1" to "1/64"). For the smallest training set sizes, mean errors increased for the stochastic search approaches (Random and Guided). This can be explained by the larger number of potential configurations causing a reduction in search efficiency. The Exhaustive version has been omitted from this experiment due to the explosion of the configuration space caused by increased granularity making exhaustive computation infeasible.

4.3.1 Comparing search configurations within DGFS

This section compares the experimental performance of the various lag scheme and constraint set configurations presented for this domain. The purpose is to show from a prediction accuracy perspective which methods perform better than the others based on the training dataset size. Figure 4.7 shows the test set accuracy of the best performing stochastic search method "Guided" against the exhaustive search method "Exhaustive" and the no feature selection "No FS" benchmark.

Statistical significance testing of all method pairs is given in Table 4.14. This comparison finds that the "No FS" benchmark does not exceed prediction accuracy of the DGFS Guided

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Greedy	32.2	32.2	32.2	32.4	33.7	35.7	39.8	67.2	72.1	61.9
Random	27.1	26.8	27.1	28.1	28.5	30.6	34.8	55.5	61.9	45.7
Guided	24.7	25.1	24.9	26.5	28.1	27.9	31.5	46.1	56.2	49.1
No FS	32.3	33.1	34.1	35.9	41.5	45.1	67.9	121.1	137.5	115.0

Table 4.10: Without constraints. Test set root-mean-square error for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain with no feature class constraints.

Table 4.11: Incorrect constraints set example for SyntheticRiver.

Variable 1	Constraint	Variable 2
dv1	BEFORE	rw2
dv1	BEFORE	dv5
dv5	BEFORE	rw4
dv1	BEFORE	rw5
dv1	BEFORE	rw4

methods for any dataset size which provides evidence that the framework's approach is valuable for improving accuracy. The Exhaustive search version consistently outperforms the other search methods as well, but the exhaustive search method relies on careful design of the feature class constraints to ensure the exhaustive search is of tractable size. Among the guided search configurations, the guided search with constraints performs best overall. However, there are several instances where the no constraints guided search has better accuracy. Overall, we find the DGFS approach improves prediction accuracy in a reliable way when the amount of training data is reduced.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Greedy	71.6	72.5	73.0	75.2	78.0	85.7	89.0	108.2	76.4	92.5
Random	28.6	30.3	29.3	31.5	35.2	33.4	41.3	67.6	71.6	57.7
Guided	25.4	26.6	26.8	27.0	29.1	29.2	34.1	62.4	54.2	56.1
No FS	32.3	33.1	34.1	35.9	41.5	45.1	67.9	121.1	137.5	115.0

Table 4.12: Incorrect constraints. Test set root-mean-square error for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain with incorrect constraints. Values are in kcfs.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Greedy	32.2	32.2	32.2	32.4	33.7	35.7	39.8	67.2	79.9	67.5
Random	27.2	27.5	27.0	28.1	29.3	32.2	36.1	57.5	66.4	51.1
Guided	25.5	25.1	24.9	26.8	28.1	28.1	34.1	49.8	54.2	47.2
No FS	32.3	33.1	34.1	35.9	41.5	45.1	67.9	121.1	137.5	115.0

Table 4.13: Expanded lag choices. Test set root-mean-square error for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain with expanded granularity lag choices. Values are in kcfs.

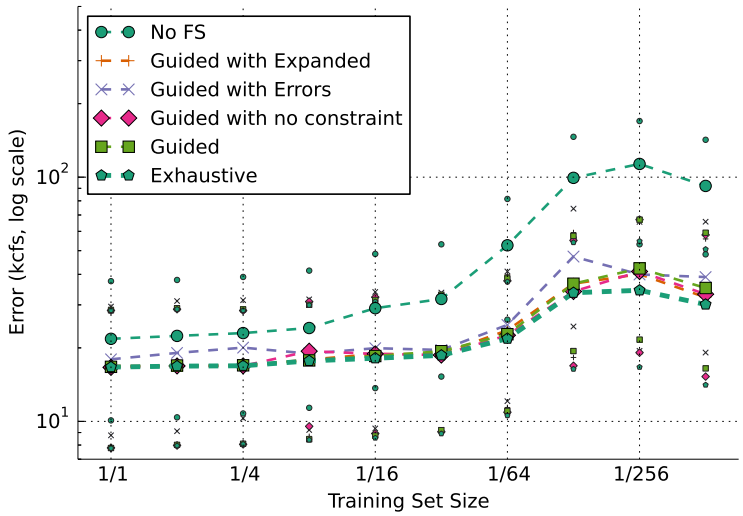


Figure 4.7: Efficiency curve for various methods on SyntheticRiver dataset: error distribution percentiles versus training set size for various feature selection approaches. Errors are presented with a logarithmic y-scale to make differences more visible across the range of error measurements.

Table 4.14: Statistical significance of reduced training set experiments using PLS for the underlying learner. Each symbol in a cell indicates that the row method is more accurate than the symbol's method and that the mean error difference is significant by the Wilcoxon Signed-rank Test with Holm-Bonferroni correction ($\alpha = 0.05$). More symbols in a cell indicate the row method dominates the performance of many other feature selection methods under comparison.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
No FS (+)										
Guided with Expanded (△)	+ ▽	+ ▽	+ ▽ ◁ ▷ ×	+ ◁	+ ▽ ◁	+ ▽ ▷	+ ▽	+ ▽	+ ▽ ◁ ▷	+ ▽ ◁ ▷
Guided with Constraint Errors (▽)	+	+	+	+	+	+	+	+	+ ◁ ▷	+
Guided with No Constraints (◁)	+ △ ▽ ▷ ×	+ ▽ ▷	+ ▽ ▷ ×	+ ▽	+ ▽	+ △ ▽ ▷	+ △ ▽ ▷	+ △ ▽ ▷	+ ▷	+ ▽ ▷
Guided (▷)	+ △ ▽	+ ▽	+ ▽	+ △ ▽ ◁	+ △ ▽ ◁	+	+ △ ▽	+ △ ▽	+	+ ▽
Exhaustive (×)	+ △ ▽ ▷	+ ▽ ◁ ▷	+ ▽ ▷	+ △ ▽ ◁ ▷	+ △ ▽ ◁ ▷	+ △ ▽ ▷	+ △ ▽ ▷	+ △ ▽ ◁ ▷	+ △ ▽ ◁ ▷	+ △ ▽ ◁ ▷

4.3.2 Analysis of Chosen Features

Another benefit of the DGFS framework is in the ability to analyze the chosen lag scheme (and model coefficient weights) for domain understanding. In this synthetic scenario, it is possible to directly compare the chosen model coefficients with the ground truth coefficients of the underlying data generation function. We show the chosen lag schemes for the "1/1" dataset size and the "1/32" dataset size to illustrate how the DGFS framework facilitates feature selection. The time lags encoded in the underlying generating function ("Ground Truth") are provided as a point of comparison.

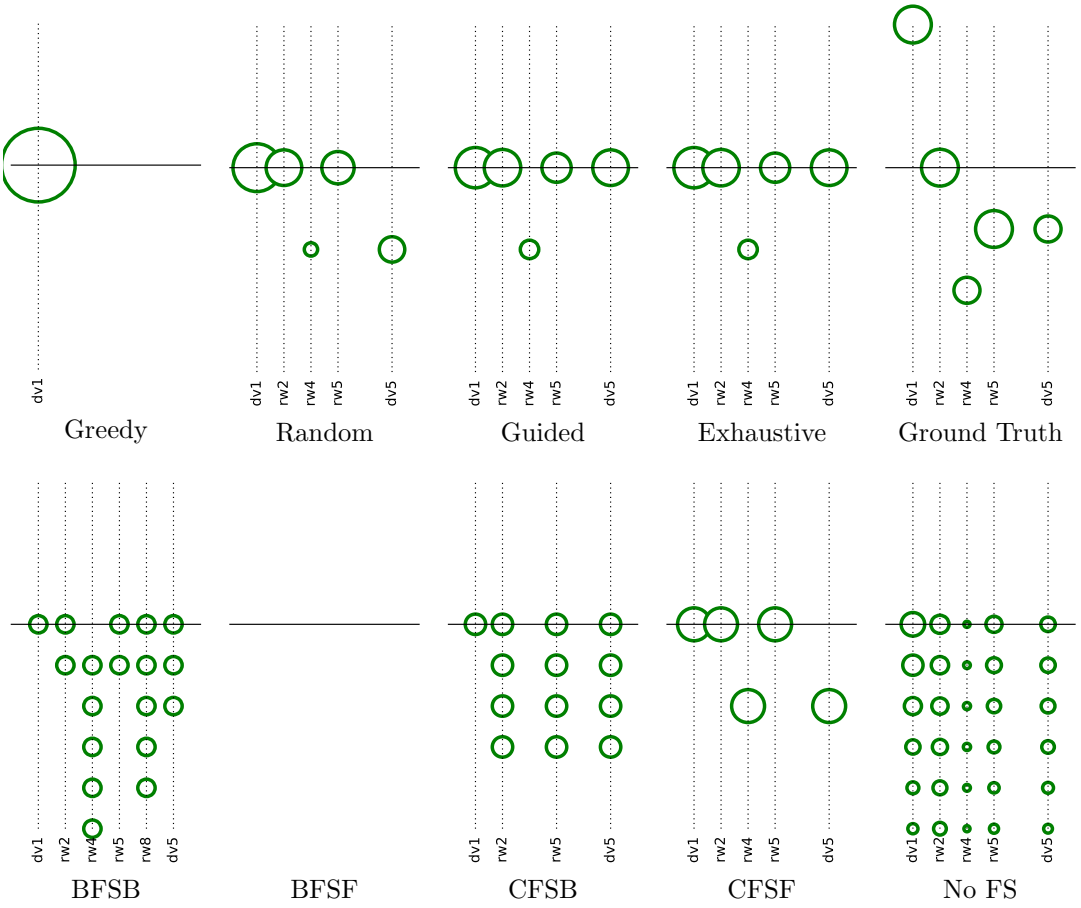


Figure 4.8: Chosen lag scheme and model coefficient magnitudes for each feature selection search method in the "1/2" dataset size.

At the "1/2" dataset size (Figure 4.8), the feature sets and coefficient weights for DGFS methods are close to the most recent value for all relevant variables. The BFSB, CFSB, and No FS methods choose a range of time lags for each variable. As there is noise in each observation, this approach of choosing several time lags for the same variable could improve prediction accuracy. However, for smaller training set sizes, choosing many time lags for the same variable can make the overall model more likely to overfit.

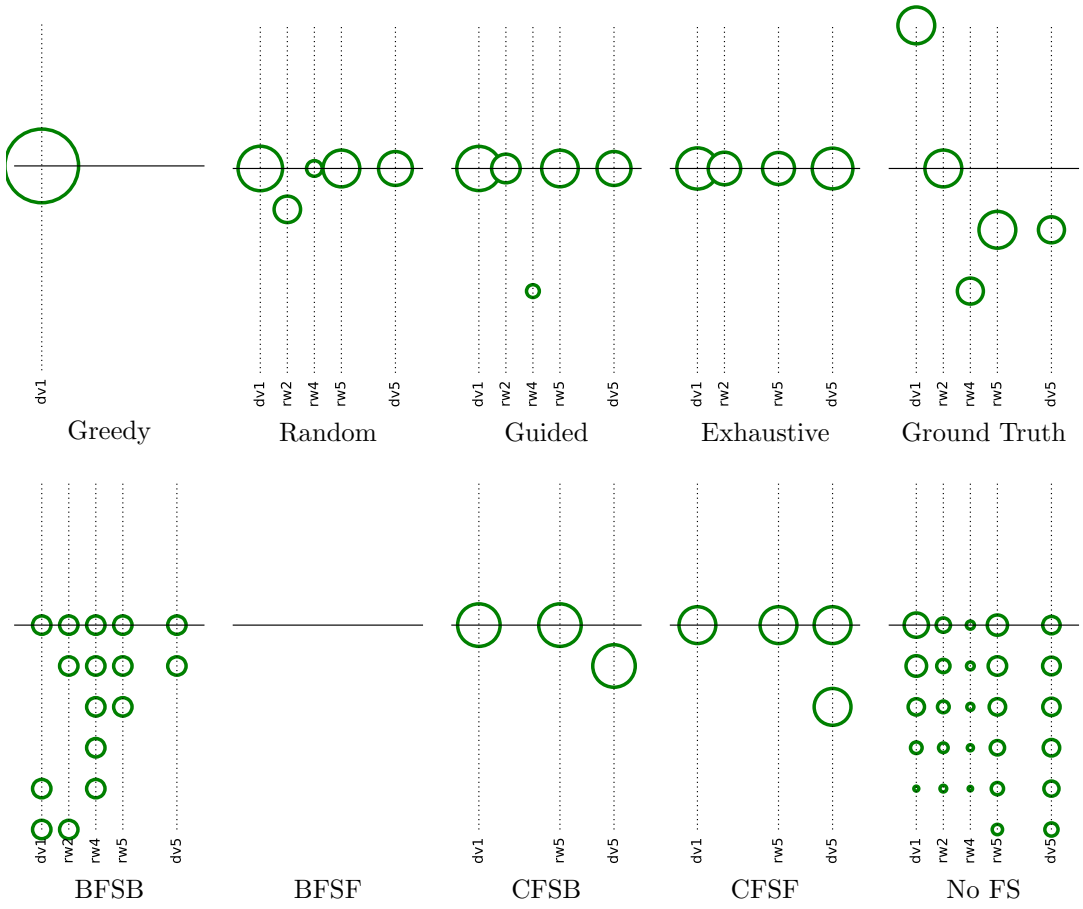


Figure 4.9: Chosen lag scheme and model coefficient magnitudes for each feature selection search method in the "1/32" dataset size.

At the "1/32" dataset size (Figure 4.9), many of the search methods produce coefficient assignments more similar to the "Ground Truth" configuration. This is particularly true for the Guided and Exhaustive DGFS search methods. Also, CFSB and CFSF produce schemes broadly similar to the ground truth. Analysis of the trained models in this way can facilitate better understanding of the domain and results.

4.4 Conclusions

These variations in performance suggest that different feature selection approaches should be chosen using the properties of the learning scenario. When training data is unlimited and the underlying concept is unchanging (corresponding to the "1/1" dataset size), an explicit feature selection strategy may not be necessary. If less data is available for training or the concept is changing, calibration using all available variables may produce models with larger errors. Using a feature selection method can alleviate the increase in errors to some extent. But as these experiments show, a feature selection search with a bias (incorporating known constraints between variables) can reduce the errors further.

It is a common circumstance for a model to be calibrated in scenarios with little available training data. This can be due to the expense or difficulty of obtaining data or because valid data is scarce due to rapid changes in the underlying concept. Our proposed method, DGFS, provides candidate solutions for this situation.

Experimentation based on synthetic data allows for a high degree over control over the phenomena being tested. The experiments in this chapter provide several important contributions to understanding feature selection in multivariate sequential data:

1. The Multi-Concept experiments (Section 4.2) show the effect of the model testing configuration choice (a.k.a. the dataset "splitting" method) on accuracy measurements in machine learning on sequential data. When the dataset does not obey the *i.i.d.* property, some experiment configurations from the literature may leak information between the training and test sets. It is important to use a method which avoids information leakage to obtain accuracy measurements that are representative of performance for as-yet-unseen samples.

2. The Multi-Concept data is also employed to demonstrate the effects of irrelevant variables on multivariate machine learning. Intuitively, irrelevant variables should have no benefit for prediction. But in practice, irrelevant variables are correlated to relevant variables so they can increase accuracy of the prediction model.

3. The DGFS approach significantly contributes to improvements in model accuracy when training set size is reduced. The SyntheticRiver domain example demonstrates the effect of training set size reduction on accuracy. In addition, the effect of incorrect constraint choices is examined, and it is found that incorrect constraints can reduce empirical model accuracy. Also, this chapter examine the effect of improved granularity in lag choices.

4. By examining the chosen feature selection and model coefficient values which are output by the feature selection search process, domain knowledge can be extracted as well.

The following three chapters will examine DGFS in the context of prediction in real-world and complex application domains where the underlying phenomena being predicted is not fully observable.

Chapter 5

Applied Domain: Airline Ticket Price Prediction

Cost minimization by strategic timing of airline ticket purchasing is a challenging prediction scenario that can leverage both historical price data and domain knowledge. To address this problem, we introduce an algorithm which optimizes purchase timing on behalf of customers and which provides performance estimates of its computed action policy as predictions are made. Given a desired flight route and travel date, the algorithm uses machine learning methods on recent ticket price quotes from many competing airlines to predict the future expected minimum price of all available flights. The main novelty of our algorithm lies in using a systematic feature selection technique, which captures time dependencies in the data by using time-delayed features, and reduces the number of features by imposing a class hierarchy among the raw features and by pruning the features based on in-situ performance. Our algorithm achieves much closer to the optimal purchase policy than other existing decision theoretic approaches for this domain, and meets or exceeds the performance of existing feature selection methods from the literature.

5.1 Introduction

The conventional wisdom for airline ticket purchasing states that it is generally best to buy a ticket as early as possible to avoid the risk of price increase. As prices do generally increase dramatically before a flight's departure, this seems correct. However, in practice the earliest purchase strategy only occasionally achieves the lowest cost ticket. This paper proposes a model for estimating the optimal time to buy airline tickets. The ultimate application is to autonomously make daily purchase decisions on behalf of airline ticket buyers to lower their

costs.

This kind of optimal airline ticket purchasing problem from the consumer's perspective is challenging primarily because prices can vary significantly on a daily basis, but consumers do not have sufficient information about pricing of specific routes and airlines. Prices do vary in this domain for several reasons. Sellers (airlines) make significant long term investments in fixed infrastructure (airports, repair facilities), hardware (planes), and route contracts. The specific details of these long term decisions are intended to roughly match the expected demand but often do not match it exactly. Dynamic setting of prices is the mechanism that airlines use to synchronize their individual supply and demand in order to attain the greatest revenue.

We assume buyers are seeking the lowest price for their ticket, while sellers are seeking to keep overall revenue as high as possible to maximize profit. Simultaneously, each seller must consider the price movements of its competitors to ensure that its prices remain competitive to achieve sufficient (but not too high) demand. Both types of relationships (buyers to sellers and airline to airline) need to be considered to effectively optimize decision making from the buyer's point of view.

A central challenge in airline ticket purchasing is in overcoming the information asymmetry that exists between buyers and sellers. Airlines can leverage historical data to predict the future demand for each flight. Demand for a specific flight is likely to change over time and will also change due to the pricing strategy adopted by the airline. For buyers without access to historical price information, it is generally best to buy far in advance of a flight's departure. However, this is not always best since airlines will adjust prices downward if they want to increase sales.

Given a corpus of historical data, we propose a learning approach which computes policies that do much better than the earliest purchase strategy. The success of our method depends on several novel contributions:

1. We leverage a developer-provided hierarchical structure of the features in the domain and use automated methods to select which features to include or prune. This technique, which we call *Developer-Guided Feature Selection* [1], computes a feature set that is more informative than existing feature selection methods. The knowledge of precedence relationships between features is provided by the practitioner when building a prediction model. For a motivating discussion on the benefits of Developer-Guided Feature Selection, see [Groves, 2013].
2. We capture temporal trends in the model by allowing time-delayed observations to supplement or replace the most recently observed value for each of the selected features. We call the time-delayed observations *lagged features*.

[1] In our previous work [Groves and Gini, 2013b], this method is called User-Guided Feature Selection. The change more accurately reflects the source of knowledge used in the bias.

We demonstrate experimentally that our model performs better than the Bing Travel "Fare Predictor", the best commercial system currently available for airline fare prediction. In addition to airline ticket price prediction, our approach is applicable to many real-world domains with properties such as the need to handle advance reservations, a range of customer values for the same product, or stock perishable inventory [Elmaghraby and Keskinocak, 2003]. Industries with these properties include hotel booking, railroad transport car rental, and broadcasting industries.

This paper extends our previous work [Groves and Gini, 2013b]. We provide an analysis of how airlines use competitive pricing and show how the market competitiveness on each airline route explains the difference in prediction performance of our model across routes. We extend our prediction method to handle specific customer's preferences, such as purchase only non-stop flights and from a specific airline. We present a sensitivity analysis of how the hierarchical structure provided by the practitioner affects prediction performance. Further, we analyze the optimal purchase timing accuracy of different models.

5.2 Background and Related Work

Briefly, we review the background on airline pricing to provide context for our work. We also outline how our work differs from existing work and how our method for feature selection guided by the modeler compares with existing feature selection methods.

5.2.1 Airline Ticket Pricing

Airline Yield Management and Dynamic Pricing

Airlines determine the prices to offer for each flight through a process called yield management which is designed to maximize revenue given constraints such as capacity and future demand estimates (for an overview, see [Belobaba, 1987; Smith et al., 1992]). Mismatches between airplane size and passenger demand are equalized through pricing, which can adjust demand. Choosing optimal pricing on an entire airline network is complex because there are instances (in hub-and-spoke networks) when sacrificing revenue on a particular flight can increase overall revenue of the entire network.

The current state of yield management in the airline industry is a direct result of decisions made about regulation in the industry [Smith et al., 1992]. Due to regulatory changes in 1979, airlines became free to adjust the price for each seat without restriction. This allowed airlines to divide the seats for each flight into different "fare classes" and charge different prices for effectively the same service.

In traditional yield management, the lowest air ticket prices quoted to customers are based on the available seats in each fare class for a particular flight (or origin-destination pair in the case of multi-stop itineraries). An airline can adjust the rate-of-fill for a particular flight by moving seats between fare classes (e.g. by moving high cost seats into lower cost fare classes). These decisions are traditionally made by humans who take into account previous demand, current sales, and competition.

Low Cost Airlines

The airline market has changed with the introduction of low cost airlines (LCAs). A study of the pricing strategies developed by LCAs in the European air travel market [Piga and Filippi, 2002] reports that tickets purchased between 30 and 8 days prior to departure are more expensive than tickets bought in other periods. Tickets bought within a few days from departure can be significantly cheaper but are not always available due to demand. LCAs do not compete against conventional airlines on price alone. They also use horizontal product differentiation to minimize the necessity to compete on price. Specifically, LCAs use secondary airports (not significantly served by conventional airlines) and fly on schedules that are maximally distant from existing players.

A later investigation [Bachis and Piga, 2011] shows that airlines that have a significant share of the traffic at an individual airport tend to have higher prices than other carriers at the same airport. Also, an airline having a large portion of traffic between two airports (one direct route) tends to have greater market power than an airline having a large portion of traffic between two airports without a direct route. There is greater substitutability on routes with one or more stops, so market power is lower. We show how market competition on individual routes affects the performance of our prediction models (Section 5.5.7).

Strategic Sales and Game Theory in Pricing

There are several efforts in the game theory community to model aspects of the airline ticket domain, usually for the purpose of understanding competitive market dynamics of the oligopoly of sellers. In [Subramanian et al., 1999], a dynamic programming model is presented for determining optimal fare class allocation (of four fare classes) on a single departure date and flight number. Valuable insights provided by this study are that booking limits do not need to change monotonically over time (may increase or decrease) and it may be better for an airline to sell a lower fare class instead of a higher fare class, for instance, due to differences in cancellation characteristics. We show an example later in Figure 5.4.

A game theory model of dynamic pricing for an oligopoly of sellers facing strategic customers, i.e. buyers who will delay purchase if there is a high likelihood of lower prices later, is presented

in [Levin et al., 2009]. The work assumes perfect foresight, all parties (sellers and customers) can estimate perfectly future outcome probabilities and utilities. If even a portion of the population of buyers is strategic, revenue for the sellers is reduced and no strategic defenses can fully ameliorate this effect. The critical conclusion of this work is that the most effective method to inhibit the impact of strategic consumers is to reduce the amount of information available to buyers. This may explain in part why, in spite of the technical feasibility, few predictive tools are available to individual purchasers of airline tickets. It is important to note that we model competitive aspects only indirectly through price relationships and not by modeling capacity, demand, and inventory as advocated by a game theory approach.

Beyond information asymmetry, airlines also use pricing behavior to reduce the effectiveness of strategic behavior. The presence of strategic consumers has been cited as a factor in increased price volatility: high price volatility makes consumers less able to judge the "right" price, and price volatility makes consumers less sensitive to price increases. The psychological effects are clear, but [Mantin and Gillen, 2011] also indicate that some systematic behaviors, such as increased price volatility, can signal price changes. Those are the phenomena which we seek to leverage in our price predictions.

5.2.2 Optimal Purchase Timing

Our work has been inspired by [Etzioni et al., 2003], where several purchasing agents attempt to predict the optimal purchase time of an airline ticket for a specific flight. The agents determine the optimal purchase time within the last 21 days prior to departure for a specific flight in their collected data set. The authors compute the purchasing policy (a sequence of wait/buy signals) for many unique simulated passengers with a specific target airline, target flight, and date of departure. The optimal policy (the sequence of buy/wait signals that leads to the lowest possible ticket price) is used as a benchmark for each simulated passenger and the cost of each alternative purchasing agent is computed. The aggregate result shows that, given these purchasing criteria, it is possible to save significantly. We understand that Bing Travel's "Fare Predictor" is a commercialized version of the models in [Etzioni et al., 2003], so we use it as a benchmark in our experimental results (Section 5.5.2).

Our work is different because we model the aggregate cost of all the flights on any airline meeting the date of travel and origin-destination pair requirements. We also allow for some preference criteria, such as number of stops and choice of airline. We model purchases up to 60 days before departure (instead of 21), we compare our results against realistic financial benchmarks (including buying as early as possible). In addition, our model provides a regression estimate for the expected best price between the current and the departure date.

5.2.3 Feature Selection

From a technical perspective, this prediction problem we address can be phrased as a machine learning problem involving both many features (possible variables which are relevant to prediction) and temporal trends. For an overview of automated feature selection techniques, see [Guyon and Elisseeff, 2003; Hall, 1999; Zhao and Liu, 2011]. There are many data-driven feature selection techniques applicable to this domain [Molina et al., 2002]. Correlation-based Feature Selection (CFS) Hall [2000] performs a filter-based feature selection using normalized Pearson's Correlation. The algorithm starts with an empty feature set and uses forward best-first search to incrementally add features. Wrapper-based methods employing search, such as best first search (BFS), coupled with a machine learning algorithm have also been employed [Kohavi and John, 1997]. Both techniques are included in our results for benchmarking.

In the context of the feature selection literature, our Developer-Guided Feature Selection is classified as a wrapper-based approach because it uses prediction performance of the underlying machine learning model measured on many feature subsets to find a well-performing feature set, The feature selection bias induced by the constraints is a way of addressing the bias-variance tradeoff when building prediction models from data [Hastie et al., 2001]. Beyond feature selection, using prior domain knowledge to guide model calibration is also an emerging approach used for improving learning of Bayesian network structures from data (e.g. [Zhou et al., 2014]).

5.3 Data Sources

The data for our analysis were collected as daily price quotes from a major travel search web site between Feb. 22 and Jun. 10, 2011 (109 observation days) for 7 different origin-destination pairs[2] for a total of 23.5 million price quotes. The specific pairs were selected to include major cities in different parts of the US and some international destinations. A web spider was used to query for each pair of origin-destination and departure date, so the results are representative of what a customer could observe.

We split this set sequentially into 3 datasets with the following lengths: 48, 20, and 41 days. The three periods are utilized as the training set, calibration set, and test set, respectively. These values were chosen so that the calibration set had at least 3 complete departures of each type (Monday and Thursday), the test set had at least 6 departures of each type, and the remainder forms the training set. Each query returned, on average, 1,200 unique round-trip itineraries from all airlines; most queries returned results from more than 10 airlines. Example queries for

[2]The 7 routes are the following origin-destination pairs: HOU-NYC, MSP-NYC, NYC-CDG, NYC-CHI, NYC-HKG, NYC-MSP, and SEA-IAD. The airport codes for the cities are: HOU=Houston, TX, USA ; MSP=Minneapolis, MN, USA, NYC = New York, USA; CDG = Paris, France; CHI = Chicago, USA; HKG = Hong Kong, China; SEA = Seattle, WA, USA; IAD = Washington, DC, USA.

individual routes and dates are in Table 5.1. Queries were made at the same time each day for consistency.

Table 5.1: Airline price quote specifications for all airlines for specific 5-day round trips. The exact dates and cities shown are for illustration purposes only.

	Example 1	Example 2
Quote Date:	13 May 2011	13 May 2011
Origin City:	SEA	NYC
Destination City:	IAD	LAX
Departure Date:	20 May 2011	20 May 2011
Return Date:	25 May 2011	25 May 2011
No. of itineraries returned:	1135	1304
No. of airlines quoting itineraries:	9	13

Bing Travel, a popular travel search web site, has a "Fare Predictor" tool that provides a daily buy/wait policy recommendation for many routes and departure dates. We obtained these recommendations daily from the site for the test set period. Those recommendations are directly compared with our results in Section 5.5.

5.3.1 Pricing Patterns in Historical Data

There are strong cyclic patterns in the time series of prices. For example, Figure 5.1 shows the mean lowest price quoted by all airlines for a specific origin-destination pair for 2 months of 5-day round trip itineraries departing on (a) Thursdays and (b) Mondays. The Thursday to Tuesday itinerary time series shows a regular drop in prices for Tuesday, Wednesday, and Thursday purchases (*days-to-departure* modulo $7 \in \{0, 1, 2\}$), while the (b) series shows significant increases for Thursday, Friday, and Saturday purchases. As expected, both exhibit price increases in the last few days before departure (*days-to-departure* ≤ 7) but series (b) exhibits this increase earlier in the time series. We posit that the majority of business flights would be Monday to Friday itineraries, and thus demand for series (b) flights would be less sensitive to price increases than leisure flights. The weekly depression in costs in (a) may be due to market segmentation: customers buying mid-week are more price sensitive than weekend purchasers. Previous studies of airline pricing confirm a similar "weekend effect" in pricing [Mantin and Koo, 2010; Puller and Taylor, 2012].

The pricing behaviors exhibited for other origin-destination pairs also differ. A high traffic origin-destination pair such as the New York City to Los Angeles route (shown in Figure 5.2) exhibits much weaker cyclic patterns. Strategic pricing is likely to have a much greater observed

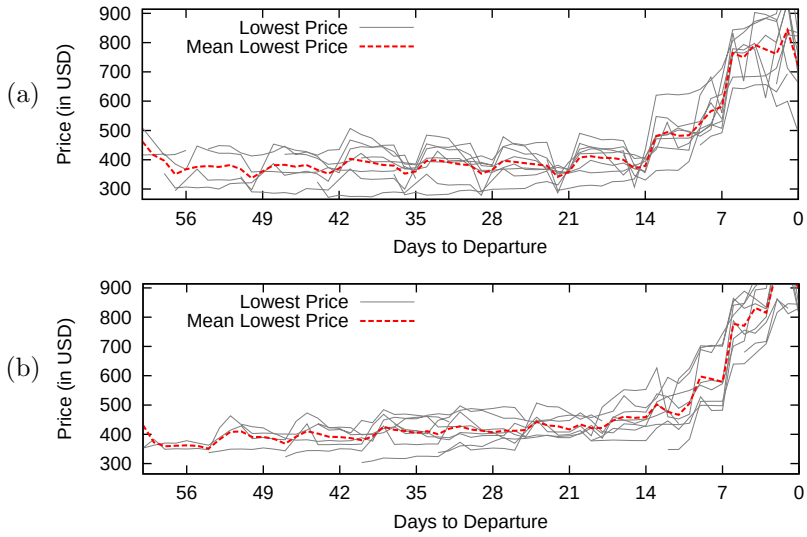


Figure 5.1: Mean lowest price from all airlines for New York City (NYC) to Minneapolis (MSP) 5-day round trip flights having (a) Thursday departure and Tuesday return or (b) Monday departure and Friday return itineraries. Each solid line series indicates the minimum price from all airlines on each query day for each departure date (8 departure dates in each graph). A dotted series indicates the mean.

effect for routes that have relatively few (2 or 3) competing airlines than for routes with a large number (> 3) of competitors [Vowles, 2000].

5.3.2 Observed Pricing Relationships

In this section, we characterize the pricing relationships between airlines that are observed in the pricing data we collected. The prices quoted each day for a specific query (such as the two examples in Table 1) often vary significantly by airline, but the prices observed have structural relationships which can be leveraged for prediction. Figure 5.3 plots the minimum price time series for each airline from four weeks of data for a specific itinerary: depart Minneapolis (MSP) on May 5, 2011, and return from New York (NYC) on May 10, 2011. Pricing patterns for competing airlines have been covered empirically in the literature (e.g. [Bachis and Piga, 2011]), and the figure illustrates these relationships.

Airlines can be clustered into several categories according to their observed pricing strategies [Obeng and Sakano, 2012]: low cost airlines ("Low" category) and "legacy" airlines ("Mid" and "High" categories). Low category airlines use their primary advantage, the ability to offer lower ticket prices due to lower internal costs, to compete aggressively against legacy airlines. Legacy airlines use other benefits to compete against Low category carriers, such as greater

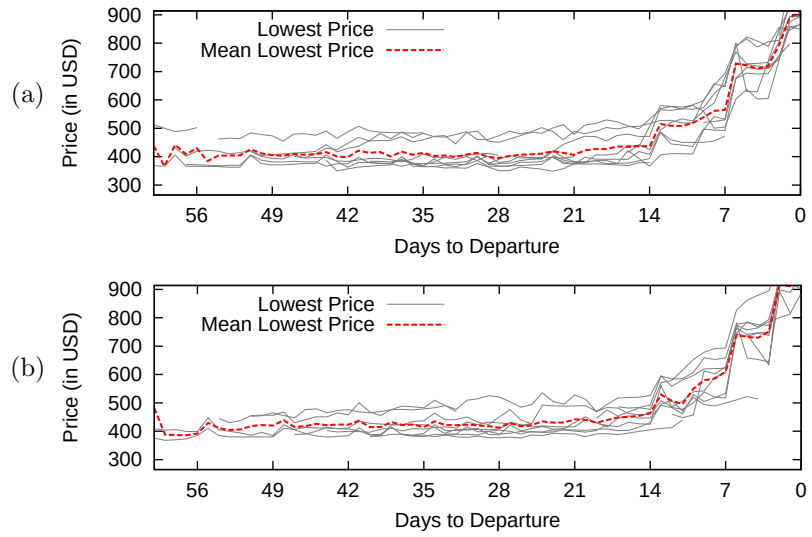


Figure 5.2: Mean lowest price offered by all airlines for New York City (NYC) to Los Angeles (LAX) 5-day round trip flights having (a) Thursday departure and Tuesday return, or (b) Monday departure and Friday return itineraries.

availability of departure times, a larger network of connecting flights, and loyalty rewards programs [Francis et al., 2006].

For a specific route's prices (shown in Figure 5.3), the competing airlines can be divided into one of three categories based on the pricing behavior. Low category airlines (referred to low cost airlines in the literature) tend to always compete on price and will consistently offer prices at or below all other types. Such airlines may even compete against other Low category carriers by lowering prices further in order to increase demand for the product. Mid level airlines are legacy airlines that tend to price aggressively for the route but will rarely set prices below the best Low category price. High category airlines are legacy airlines that do not compete aggressively based on price but will still quote (usually higher) prices for the route. Customers may still buy from Mid or High categories because of the other benefits of the specific airline or itinerary.

In this route, the airlines can be categorized based on price behavior as:

- Low category: Frontier (F9), AirTran (FL), Sun Country (SY)
- Mid category: Delta (DL)
- High category: American (AA), Continental (CO)

Some airlines will quote itineraries with a different number of intermediate stops. For example, Delta airlines (DL) quotes itineraries with no intermediate stops (non-stop, as DL-0), 1

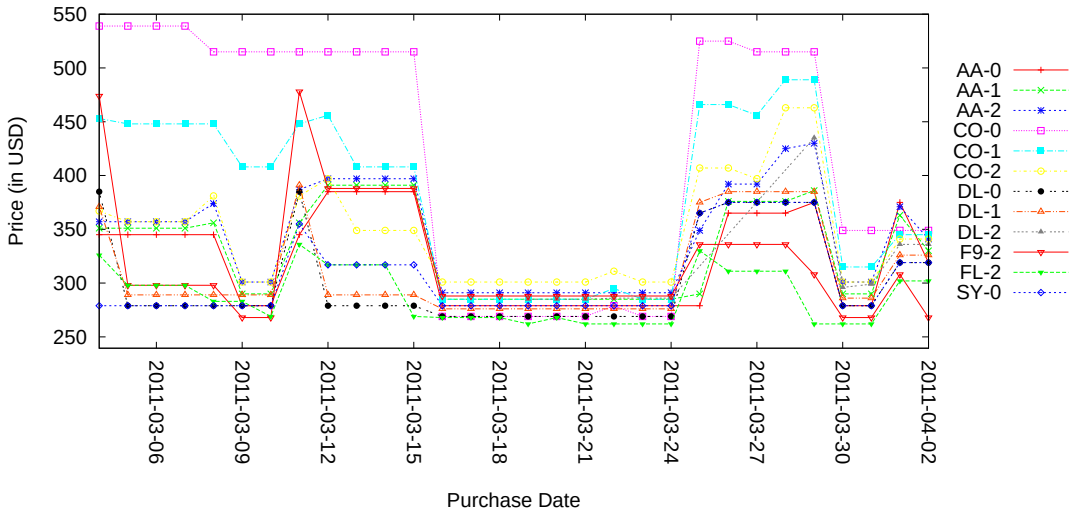


Figure 5.3: A price time series for many airlines quoting prices for the NYC-MSP route for Thursday departure (May 05, 2011) with Tuesday return. Each series represents the minimum price of the day's quotes for a specific airline and number of stops pair: for example, **DL-0** refers to all Delta Airlines non-stop flights and **AA-2** refers to all American Airlines, 2 intermediate stop flights.

intermediate stop (DL-1), or 2 intermediate stops (DL-2). For the purposes of the analysis, we treat each itinerary type separately, as shown in Figure 5.3.

The price time series in Figure 5.3 shows repeated patterns among the airlines for this departure and route. The Low category airlines are consistently at the bottom of the price range and there is a cluster of Mid-level airlines that have similar prices during periods of price stability. To characterize some of the price shifts, Figure 5.4 provides some stylized examples. In 5.4(a), the Mid-level cluster lowers its price below the prevailing Low category price; on the next day, the Low category adjusts its prices to match or beat the best price. In 5.4(b), a Low category airline lowers its price, but no other airline follows. In 5.4(c), the Low category raises its price and the Mid and High category airlines adjust their prices upward as well.

These categorizations can be made from a statistical analysis of the data. These relationships between price behavior of airlines can be leveraged using a regression model because, within a route, the relationships are persistent. A machine learning model, such as those described later in Section 5.4.3, can leverage these relationships to make predictions about future prices.

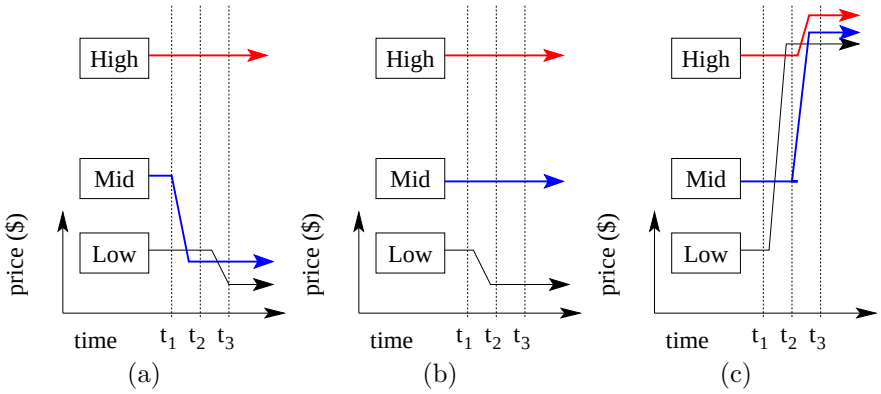


Figure 5.4: Stylized diagrams of changes in price equilibria. Three price behavior types are shown: "Low" refers to low cost airlines, "Mid" refers to medium cost airlines, and "High" refers to high cost airlines.

5.4 Proposed Model

When constructing prediction models for real-world domains, practical complexities must be addressed to achieve good prediction results. Typically, there are too many sources of data (features). Limiting the set of features in the prediction model is essential for good performance, but prediction accuracy can be lost if relevant inputs are pruned. This is even more critical when the number of observations available is limited, as often occurs in real-world domains.

To meet this challenge, we construct a prediction model (Figure 5.5) that involves the following distinct steps, which we then describe in detail:

1. Feature Extraction and Feature Class Constraints – The raw data observed in the market are aggregated into a fixed length feature set.
2. Feature Selection and Lagged Features – A lag scheme is computed using a hierarchy of the features that incorporates some domain knowledge.
3. Regression Model Construction – Using the augmented feature set generated from the lag scheme, a regression model is generated, for instance using partial least squares (PLS) regression.
4. Policy Computation – A search of decision threshold parameters is done to minimize cost on the calibration set.
5. Optimal Model Selection – For each candidate model computed using the previous steps, the one which performs best on the calibration set is chosen. The final performance is estimated on the test set.

Figure 5.5 shows the distinct components of the model. Information used in the model

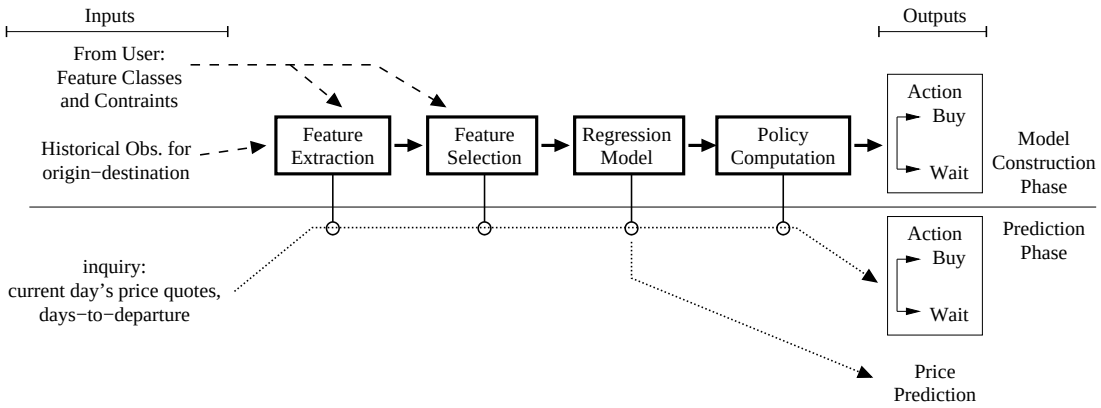


Figure 5.5: Airline prediction model components, input data, and output

construction phase includes historical data from the origin-destination pair and domain knowledge. The prediction phase for computing a buy/wait action at each decision day in the test set generates the action sequences scored in our experimental results.

5.4.1 Feature Extraction and Feature Class Constraints

The large number of itineraries (1000s) in each daily query made some data aggregation necessary. The features extracted are aggregated variables computed from the large list of prices from individual query days. For each query day, there are often many airlines quoting flights for that specific origin-destination and date combination. Airlines can vary their prices due to strategic decisions or to changes in available capacity.

To ensure a consistent, informative feature set, we build separate features for airlines quoting prices for 40% or more of the quote days in the dataset. For each airline exceeding the 40% criteria, its quotes are subdivided based on the number of intermediate stops: 0-stops ("nonstop"), 1-stop, or 2-stops. For each subdivision, three aggregate features are computed: the minimum price, mean price, and the number of quotes (corresponds to the EACH-SEPARATE feature class in Table 5.2). These three aggregates are computed for the set of all the quotes from each airline (corresponds to the EACH-AGGREGATE feature class). So for each airline, 12 features are computed on each quote day. For airlines not exceeding the 40% criteria, their itineraries are combined into a separate "OTHER" category placeholder. Finally, these same 12 aggregates are generated for all quotes and are placed in the "ALL" airlines category (corresponds to the ALL-STOPS and ALL-AGGREGATE feature classes). Boolean variables are added to indicate the query's weekday (for instance, "Quote DoW is Mon" is true if the quote is retrieved on Monday). A *days-to-departure* (number of days between the query and departure dates) value is computed based on the departure date.

Concurrent with the feature extraction process (when converting the 1000s of itineraries for each query into feature values) is the computation of the target variable's value (e.g. minimum of ALL-min-A or minimum of DL-min-0). For example, in the computation of the DL-min-0 target variable on each query day, the minimum price of all non-stop tickets on all queries between the query date and the departure date is the value assigned for that observation day. This value represents the minimum possible cost ticket that could be purchased to satisfy the travel need given the preferences (DL=Delta Airlines, 0=non-stop only flights) and departure date.

Table 5.2: Raw features by feature class for each quote day for a specific departure day and route.

Feature Class	Variable Count	Variable List
DETERMINISTIC	8 vars	Days-to-departure, Quote DoW is Mon, Quote DoW is Tue, ..., Quote DoW is Sun
ALL-AGGREGATE	3 vars	ALL-min-A, ALL-mean-A, ALL-count-A
ALL-STOPS	9 vars	ALL-min-0, ALL-mean-0, ALL-count-0, ALL-min-1, ALL-mean-1, ALL-count-1, ALL-min-2, ALL-mean-2, ALL-count-2
EACH-AGGREGATE	18 vars	DL-min-A, DL-mean-A, DL-count-A, ..., OTHER-min-A, OTHER-mean-A, OTHER-count-A
EACH-STOPS	54 vars	DL-min-0, DL-mean-0, DL-count-0, DL-min-1, DL-mean-1, DL-count-1, DL-min-2, DL-mean-2, ...

[1]Note: The number of variables in some classes (EACH-AGGREGATE, EACH-STOPS) will vary based on the number of airlines quoting the route. The counts given are specific to the NYC-MSP route (92 total raw variables). Variables are named as "<airline>-<statistic>-<#ofStops>": e.g. ALL-min-A = minimum price quoted by any airline, ALL-min-0 = minimum price quoted by any airline for non-stop flights only, DL-min-A = minimum price quoted by a specific airline (DL = Delta Airlines). Variables named like "Quote DoW is Mon" are Boolean variables indicating the weekday the quote is retrieved.

A list of the features for a specific departure day and origin-destination for each query day is shown in Table 5.2. Each of the 92 features is in a feature class based on its specificity using the feature class hierarchy in Figure 5.6. The number of variables in some classes (EACH-AGGREGATE, EACH-STOPS) will vary based on the number of airlines quoting the route. Variables are named with the compact scheme "<airline>-<statistic>-<#ofStops>": e.g. ALL-min-A = minimum price quoted by any airline, ALL-min-0 = minimum price quoted by any airline for non-stop flights only, DL-min-A = minimum price quoted by a specific airline (DL = Delta Airlines).

The feature classes are groupings of raw features in the domain. For this domain, there is a deterministic set of features ("DETERMINISTIC") which includes information about the number of days from the quote day until departure, and the 7 binary variables indicating the weekday of the quote (i.e. Monday, Tuesday, ...). At most one instance of the DETERMINISTIC feature class is included because it contains deterministic information. Knowing the value for one day

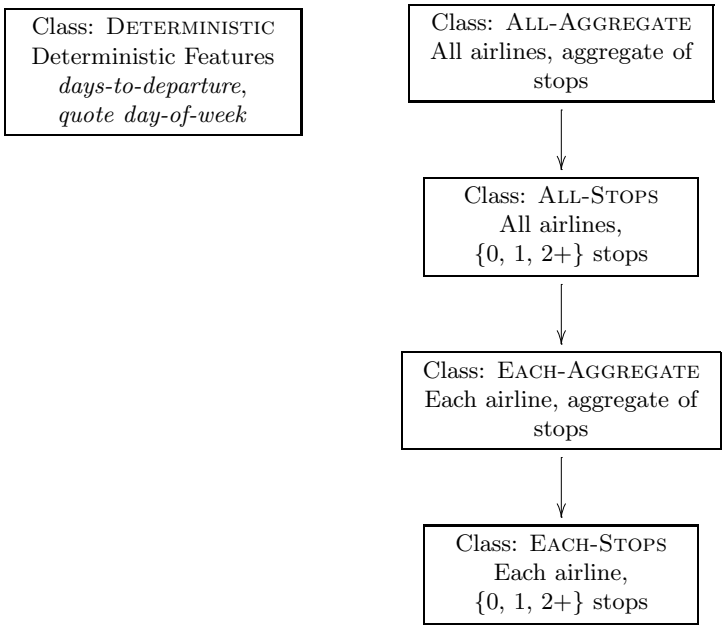


Figure 5.6: Lag scheme class hierarchy for price prediction. Arrow denotes a *subset* relationship (e.g. class ALL-AGGREGATE should have an equal or greater set size than class ALL-STOPS).

allows for all time lagged day's values to be determined exactly, so there is no informational benefit to include more than one instance.

The ALL-AGGREGATE feature class contains raw features applicable to all airlines: the minimum price, mean price, and number of price quotes from any airline on the quote day for the origin-destination. The ALL-STOPS feature class is similar to ALL-AGGREGATE but has separate statistics based on the number of intermediate stops. This class is "more specific" than ALL-AGGREGATE, the data are more fine grained, so we say that ALL-STOPS is more specific than ALL-AGGREGATE. This is indicated with a directed edge between the two in Figure 5.6. ALL-STOPS also contains more variables than ALL-AGGREGATE (9 vs. 3). Using similar arguments the two more specific classes EACH-AGGREGATE and EACH-STOPS are constructed. The size of EACH-AGGREGATE and EACH-STOPS (in terms of the number of significant competitors) will vary based on the origin-destination to be predicted. For the NYC-MSP route, there are five major competitors so there are 18 features in EACH-AGGREGATE[3]. For this domain, we constrain the feature classes to always include the more general classes before more specific classes using the precedence hierarchy shown in Figure 5.6. Domain knowledge is used to build

[3] 5 major competitors plus the "other" category for small airlines and 3 features per airline: $(5 + 1) * 3 = 18$.

this structure. These precedence relationships prevent highly specific and possibly redundant information from being included in the augmented feature set before more general variables are included.

After feature extraction and identification of feature classes (shown in Table 5.2), known hierarchical relationships between the feature classes will make up the constraint set. For the airline domain, the feature class relationships in Figure 5.6 are common sense domain knowledge that can be elicited from even a novice practitioner with basic knowledge of the domain to bias the feature selection search. For a discussion about alternative constraint sets, see section 5.5.5.

5.4.2 Feature Selection and Lagged Features

Using only the most recent values (92 features for the NYC → MSP route) as the entire feature set may provide reasonable prediction results in some domains, but such a model cannot predict trends or temporal relationships present in the data. This simple scheme is shown in Table 5.3(b) as *Most Recent Observations*.

Table 5.3: Basic lag schemes used for benchmarking of feature selection methods. The dots "•" indicate that the feature class at the corresponding time lag is included in the feature set provided to the learning model.

(a) Full Lag Scheme (lags are in days)

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A	•	•	•	•	•	•	•	•
ALL-S	•	•	•	•	•	•	•	•
EACH-A	•	•	•	•	•	•	•	•
EACH-S	•	•	•	•	•	•	•	•

(b) Most Recent Observations

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A	•							
ALL-S	•							
EACH-A	•							
EACH-S	•							

The need to represent temporally-offset relationships (such as weekly cycles or trends) motivates adding time-delayed observations to the feature set as well. This is accomplished by including past days' observed values in addition to the current day's value in the feature vector used for learning. We refer to this as the addition of *lagged features*. For instance, if the cost of a route on day $t-7$ is representative of the price available on day $t+1$, the 7 day delayed observation should have a high weight in the model. A regression model which includes all time-delayed instances up to a depth of n days of all features can produce good results, but the

inclusion of too many variables into the model can result in poor performance. A diagram of this model is in Table 5.3(a). The performance of both *Full Lag Scheme* with ($n = 8$) and *Most Recent Observations* is shown later in Table 5.6.

Lagged features, known as tapped delay lines in engineering, are well known in the time series literature [Sauer, 1994] and have been used as input to machine learning models [Wan, 1994; Du and Swamy, 2014]. The augmented feature set can be constructed from a generated lag scheme configuration. For each feature class and lagged offset pair containing a dot (•), all variables in the class are added to the augmented feature set for the specified lag offset.[4]

Our technique assumes that more recent observations are likely to have high informational value for price prediction, but time-delayed features may hold informational value as well (i.e. the environment is not completely stochastic). The time between a change in the market and its effect on the target variable may be longer than one day. Lagged variables can leverage those delayed relationships.

By examining all combinations of feature classes we can automatically tune the feature vector to achieve better results. For each feature class, all contiguous subsets of lags and the empty set are examined as possible candidates. Another constraint is added due to relationships between classes: more specific feature classes cannot have more time lagged instances than more general classes (i.e., the more specific class' lags will be a subset of the more general class' lags).

Time-delayed observations from the target variable (such as the all airline minimum price in *Class* ALL-AGGREGATE) are likely to be most predictive because they are most general. Time-delayed observations from other more-specific feature classes *may also be* but are less likely to be predictive. The hierarchy and strict ordering of lagged data are based on this principle. By constraining the classes so that the less informative classes contribute fewer features, we prevent the inclusion of irrelevant features.

Next, time lagged data are used to form the augmented feature set. A search of all the possible lag configurations is done to find the best performing configuration for the given departure date and origin-destination. Note that the optimal configuration may be different for each date and route.

The inclusion of a little domain knowledge using feature classes and the class hierarchy reduces significantly the number of possible feature set configurations that need to be searched. Without the hierarchy and constraints between features, there are $\approx 10^{82}$ configurations of the

[4]A brief example is provided: Given a lag scheme with dots only on the ALL-A feature class, if the ALL-A feature class has dots on lagged offsets 0, -1 and -2 only, then the augmented feature vector will contain 9 augmented variables which are (variable name, time offset) tuples. These 9 augmented variables will be : (ALL-min-A,0), (ALL-mean-A,0), (ALL-count-A,0), (ALL-min-A,-1), (ALL-mean-A,-1), (ALL-count-A,-1), (ALL-min-A,-2), (ALL-mean-A,-2), (ALL-count-A,-2). The time offset refers to the number of days into the past from the quote day the value should be extracted from the observations: (ALL-min-A,0) refers to today's value for ALL-min-A, and (ALL-min-A,-1) refers to yesterday's value for ALL-min-A.

92 original features.[5] Without the constraints between classes, the number of configurations of the 4 feature classes (having time delays of $\{\varnothing, 0, 1, 2, \ldots, 7\}$) would be very large at $\approx 1.9 * 10^6$, but many configurations are uninteresting variants.[6] The number of lag schemes formulated with the hierarchy and constraints in Figure 5.6 for a maximum time delay of 7 days is 8517.[7] Using both the feature classification and the constraint hierarchy allows for "interesting" lag schemes to be efficiently evaluated in a smaller number of evaluations.

The advantage of the automated lag scheme search is that it produces results similar to what a domain expert could do but using only minimal domain knowledge. Table 5.4 shows optimal lag schemes for several routes. The results of the optimal lag scheme search can uncover some surprising relationships in the data. For example, it is interesting to note that non-stop targets in Table 5.4(b, d) benefit from a larger feature set (both in temporal depth and feature class breadth).

Before we can explain how the optimal lag scheme is selected, we need to describe the machine learning methods we use and how an action policy (buy or wait) is computed.

Counting Valid Configurations Given Constraints

The number of configurations for a given a set of constraints can be computed precisely and does not require a complete enumeration. For a chain of subset constraints among feature classes that allow contiguous subsets of lags and *null*, the number of possible configurations can be computed with the following recurrence relations:

$$A(k,n) = \begin{cases} k & \text{if } n = 1 \\ n & \text{if } k = 1 \text{ and } n > 1 \\ \binom{n+(k+2)}{n-1} + A(k-1,n) & \text{otherwise} \end{cases} \tag{5.1}$$

$$B(k,n) = \begin{cases} 1 + k & \text{if } n = 1 \\ B(k,n-1) + A(k,n) & \text{if } n > 1 \end{cases} \tag{5.2}$$

where k corresponds to the number of feature class levels, and n corresponds to the number of unique lag values for each feature class. For a chain of length four with 8 possible values $\{0, 1, 2, \ldots, 7\}$ and null for each feature class the number of configurations is the value of $B(4, 8) - 1$, where minus one is due to removing the assignment with $\varnothing$ (empty set) values for all feature classes. The result is $(8518 - 1) = 8517$.

[5] 84 price features and 8 deterministic features (days-to-departure and weekday of quote) $= (2^8) \times (2^{84*8})$

[6] For each of the 4 feature classes, there are 8 lags (plus $\varnothing$) yielding 36 contiguous subsets (plus one additional configuration for $\varnothing$). The count of possible combinations for 4 feature classes is $37^4 = 1874161$.

[7] The mathematical formula for finding the exact number of unique configurations among a chain of subset constraints (All-Aggregate to Each-Stops) is available in Section 5.4.2.

5.4.3 Machine Learning Method

Our Developer-Guided Feature Selection is agnostic about the underlying machine learning algorithm used for learning. In our experiments we used many such algorithms. Results for a sample of well studied machine learning algorithms from the literature are shown in Table 5.6 in Section 5.5.

Some of the learning methods we use are classifiers and output a buy or wait action instead of a price. In this case, the learning method and decision threshold components are replaced with a classification algorithm or an external action strategy (such as Bing Travel's recommendations). This modification is indicated in the Learning Method Output column of Table 5.6 by "Buy/Wait." The column contains "Regression" when instead a regression model is used.

As the feature selection framework considers the underlying learning model as a black box, a wide range of time-series methods could be applied using this framework as well [Box et al., 2013]. Our experiments utilize Ripper and Linear Regression univariate time-series methods as points of comparison. For the majority of our experiments, we use a machine learning regression or classification learner. The most compelling reason to use a multivariate regression/classification learner over a multivariate time-series learner is the following: the data for each route is actually made of a collection of shorter time-series. There is one time series for each departure date which has a length of at most 60 days. The frequent restarts required for standard time-series models make their application somewhat less natural. Specifically, it necessitates also choosing an initial state for the time-series model at each restart.

For pure regression-based methods we use support vector regression (nuSVR, [Schölkopf et al., 2000]), Partial Least Squares (PLS) regression, and ridge regression. The decision tree classifier we use (REPTree, [Witten and Frank, 2005]) can also predict values of continuous functions, so both modes are shown in the experiments for comparison.

Another method we utilize in our experiments is a regression algorithm which creates a linear model: PLS regression. Mathematically, PLS regression deterministically computes a linear function that maps a vector of the input features $\overrightarrow{x}$ into the output variable y (the label) using a vector of weights $\overrightarrow{w}$. Several implementations of PLS exist [de Jong, 1993; Martens and Næs, 1992], each with its own performance characteristics. We use the orthogonalized PLS implementation in [Wold et al., 1983].

We have chosen PLS over similar multivariate techniques including multiple linear regression, ridge regression [Hoerl and Kennard, 2000], and principal component regression (PCR) [Jolliffe, 1982] because of its advantages and better performance. First, PLS regression is able to handle very high-dimensional inputs because it implicitly performs dimensionality reduction from the number of inputs to the number of PLS factors. Second, the model complexity can be adjusted by changing the number of PLS factors to use in computing the regression result. These factors

are analogous to the principal component vectors used in principal component regression. The number of PLS factors determines the dimensionality of the intermediate variable space that the data is mapped to (we use a limit of 5 factors in our results). The computation time does not significantly increase for a larger number of factors but the choice can affect prediction performance: too many factors can cause over-fitting, and too few factors can cause the model to be unable to represent relationships in the data. This value is adjusted in our experiments to determine the optimal model complexity in each prediction class. Finally, the algorithm is generally robust to highly collinear or irrelevant features.

5.4.4 Policy Computation and Evaluation

An obvious approach to choosing a good regression model (lag scheme and trained machine learning model) is to use the model with the highest prediction accuracy, but this may not be the model that generates the lowest average cost policy. Instead, we rank the models by measuring the cost that results from following the computed policy recommendation. To use the regression output (an expected future price) to compute an action policy, we introduce the concept of a decision threshold function. Given $\hat{e}_t$, the model estimate future price at time t, the current observed price p_t and the current number of days-to-departure d_{dtd} (an integer), the current action policy $r_t \in \{\text{BUY}, \text{WAIT}\}$ is computed by Equation 5.3.

$$r_t = \begin{cases} \text{BUY} & : \hat{e}_t > p_t \times (c + (1/30) \times s \times d_{dtd}) \\ \text{WAIT} & : \text{otherwise} \end{cases} \tag{5.3}$$

The two parameters c and s are expressed as real numbers. Intuitively, c can be thought of as an adjustment in the likelihood of a BUY signal. Values of $c > 1.0$ correspond to a policy that is only likely to emit BUY when the current price is far below the expected future price. This situation indicates the current price is a bargain for the customer. The parameter s corresponds to the percent change in the threshold per 30 days of advance purchase (0.02 corresponds to a 2% change in the threshold at $d_{dtd} = 30$). Values of $s > 0.0$ generate a policy more likely to WAIT when far from departure date. When a departure is far in the future and $s > 0.0$, the agent is more likely to wait until a highly favorable (low) price appears before deciding to purchase. Adjusting these two parameters can be thought of as determining the optimal level of risk depending on the current price and the degree of advance purchase. The range of c and s values searched in our experiments was $[0.7, 1.3]$ and $[-0.1, 0.1]$, respectively, in increments of 0.01.

We use this two-parameter approach to make decisions, because it is simple, works well, and provides an intuitive understanding of the policy computation. This is not to rule out more sophisticated approaches, such as reinforcement learning. We leave exploration of this aspect

for future work.

5.4.5 Optimal Model Selection

The proposed search of lag schemes is exhaustive, but due to the feature class hierarchy, the number of configurations is relatively small and can be fully explored. The process is outlined in Algorithm 11.

Algorithm 11: Developer-Guided Feature Selection: training, calibration, and testing process

Data: historical price quotes for origin-destination (training set $hist_{\text{TR}}$, calibration set $hist_{\text{CA}}$, test set $hist_{\text{TE}}$), feature classes (fc) and feature class constraints (co)
Result: best observed lag scheme (ls_{best}), decision threshold parameters (s_{best}, c_{best}), calibration set mean cost ($score_{\text{CA}}$), test set mean cost ($score_{\text{TE}}$)

```
{ score_CA ← null,  ls_best ← null,  c_best ← null,  s_best ← null }
RM_TR, P_TR ← doFeatureExtraction(hist_TR, fc)
RM_CA, P_CA ← doFeatureExtraction(hist_CA, fc)
L ← generateLagSchemes(fc, co)
for ls in L do
    AM_TR ← buildAugementedFeatureMatrix(RM_TR, ls)
    PriceModel ← trainRegressionModel(AM_TR, P_TR)
    P̂_CA ← predict(PriceModel, AM_CA)
    for c in {0.7, 0.71, ..., 1.3} do
        for s in {−0.1, 0.09, ..., 0.1} do
            score ← doPolicyComputationAndScore(AM_CA, P̂_CA, P_CA, c, s)
            if score_CA = null or score_CA > score then
                {score_CA ← score,  ls_best ← ls,  c_best ← c,  s_best ← s }
RM_TE,P_TE ← doFeatureExtraction(hist_TE, fc)
AM_TE ← buildAugementedFeatureMatrix(RM_TE, ls_best)
P̂_TE ← predict(PriceModel, AM_TE)
score_TE ← doPolicyComputationAndScore(AM_TE, P̂_TE, P_TE, c_best, s_best)
```

A model is constructed for each potential lag scheme: first, for each lag scheme a pricing model is generated using the training set data, then the decision threshold parameters (c and s) are calibrated on the calibration set to discover the settings with the lowest average ticket price ($score_{\text{CA}}$). Performance ($score_{\text{TE}}$) is measured by applying the calibrated model to the test set.

Table 5.4: Optimal lag schemes of a domestic route and an international route for a 5-day trip with Monday departure. The dots "•" indicate the feature class at the corresponding time lag is included in the best performing feature set. "MC" is the average model cost, "EP" the average earliest purchase cost, and "PAO" is the mean price of the model above the optimal policy price (in %).

(a) New York → Minneapolis

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A	•	•	•	•	•	•	•	•
ALL-S		•	•	•	•	•	•	
EACH-A				•				
EACH-S								

(MC: $280, EP: $317, PAO: 4.6%)

(b) New York → Minneapolis (non-stop only)

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A	•	•	•	•	•			
ALL-S				•				
EACH-A				•				
EACH-S				•				

(MC: $365, EP: $414, PAO: 7.1%)

(c) New York → Hong Kong

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A	•	•	•	•	•	•	•	•
ALL-S								
EACH-A								
EACH-S								

(MC: $1190, EP: $1207, PAO: 3.8%)

(d) New York → Hong Kong (non-stop only)

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A	•	•	•	•	•	•	•	•
ALL-S	•	•	•	•	•	•	•	
EACH-A	•	•	•	•				
EACH-S	•	•	•	•				

(MC: $1404, EP: $1416, PAO: 6.1%)

5.5 Experimental Results

The experiments were designed to estimate real-world costs using various prediction models to develop a purchase policy. A survey of the literature revealed that airlines assume a relatively fixed rate of purchases until a flight is full, and most tickets for a flight are sold within 60 days of departure [Belobaba, 1987]. Using these facts, we measure performance as the cost of following the purchase recommendations for a specific departure once for purchases between 1 and 60 days before departure ($dtd \in \{1, 2, \ldots, 60\}$). This measure involves hypothetically purchasing an itinerary precisely 60 times for each purchase algorithm under test (but some purchases may be deferred for a few days based on the model output). Each of the 60 purchases is called a *purchase episode*. On each query day, the policy generator uses the currently observed prices (and possibly time-lagged observations) to compute a BUY or WAIT signal for the day. Using the sequence of BUY and WAIT signals it is possible to determine the costs experienced by the algorithm for each simulated purchase. Mathematically, the cost for each simulated purchase experienced by a purchase "method" (e.g. Earliest Purchase, Optimal, etc.) is computed in Equation 5.4 where the r_i's are obtained from one departure date.

$$\text{cost}_t = \begin{cases} p_t, & \text{where } r_t = \text{BUY} \\ \text{cost}_{t+1}, & \text{where } r_t = \text{WAIT} \end{cases} \tag{5.4}$$

These costs are aggregated with data from all departure dates (DD is all departure dates in the test dataset $hist_{\text{TE}}$) to compute the score for the method in Equation 5.5.

$$score_{\text{method}} = \text{mean}(\{\text{cost}_{dd,t} \mid dd \in \text{DD and } t \in T_{dd}\}) \tag{5.5}$$

T_{dd} is all time units applicable to departure date dd. By convention the last day before departure is always labeled with a BUY signal, so cost_t is always defined.

Table 5.5 shows examples of purchasing signals generated by four different policy generators (Earliest Purchase, Our Model, Optimal, and Latest Purchase) for a specific origin-destination pair and departure date. The performance of each policy generator is the mean of the cost values across all purchase episodes. By comparing purchasing signals, we can compute the percentage above the optimal cost (PAO) for each method (i.e. how far the method is above the optimal cost). The earliest purchase strategy represents the cost of buying at the first decision point in each purchase episode and will have a PAO of usually 5–30%. The optimal purchase strategy is the lowest possible cost for the purchase episode, which can be computed optimally in hindsight and represents a PAO of 0%. Well performing policy generation algorithms should do at least as well as earliest purchase on almost all purchase episodes and should achieve costs as close

to the optimal purchase as possible. Algorithms can be compared using the PAO value, well performing methods will be between earliest purchase and optimal purchase value. Note that it is possible for an algorithm to have a PAO greater than the earliest purchase strategy due to having costs that are higher than an earliest purchase, latest purchase is an example that often achieves a much higher PAO score.

Table 5.5: Actions computed by four different methods for up to 13 days before departure for a specific request (NYC-MSP, depart May 12, 2011, return May 17, 2011). The ◯ symbol indicates a "WAIT" signal for that day, and the • symbol indicates a "BUY" signal for that day. The models are compared and scored (last column) using the mean of all costs for simulated purchases. There is one simulated purchase for each day up to the departure date.

Days to departure		13	12	11	10	9	8	7	6	5	4	3	2	1	0	**Score**
Current price (in $)		258	257	257	257	257	282	292	330	298	330	330	222	469	453	
Earliest Purchase	action	•	•	•	•	•	•	•	•	•	•	•	•	•	•	**306.6**
	cost (in $)	258	257	257	257	257	282	292	330	298	330	330	222	469	453	
Our Model	action	•	◯	◯	•	•	◯	◯	◯	•	◯	◯	•	◯	•	**289.3**
	cost (in $)	258	257	257	257	257	298	298	298	298	222	222	222	453	453	
Optimal	action	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	•	◯	•	**255.0**
	cost (in $)	222	222	222	222	222	222	222	222	222	222	222	222	453	453	
Latest Purchase	action	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	•	**453.0**
	cost (in $)	453	453	453	453	453	453	453	453	453	453	453	453	453	453	

5.5.1 Model Comparison

The experiments are first categorized by the type of feature selection employed then by the underlying machine learning algorithm used. Table 5.6 shows the results of estimated costs for several purchasing policies based on purchasing 5-day (Monday or Thursday departure) round trip itineraries from NYC to MSP (∼265 simulated purchases in each test set). The table shows how costs vary based on preferences such as a customer requiring a non-stop itinerary.

Deterministic Feature Selection

The naïve strategy, called *earliest purchase*, purchases a ticket once for each day in the α day range. Its purchase episodes terminate with a purchase event on the first day of the episode. Its mean cost is equal to the mean of prices across the α day period. The lowest achievable cost, called the *optimal cost* strategy, is based on purchasing for each of the α episodes at the lowest price between the beginning of the episode and its departure date (denoted as 0% above optimal). The comparison methodology involving simulated purchases is similar to that used in [Etzioni et al., 2003]. The action policy from *Bing Travel*'s purchase recommender applied

Table 5.6: Model results comparison on a single route for the lowest cost itinerary on any airline. All itineraries are 5 days (Monday to Friday, or Thursday to Tuesday). Cities are NYC (New York City) and MSP (Minneapolis, MN, USA). Note: "PAO" (% above optimal policy cost) for each method is computed as: $([\text{score}_{method} - \text{score}_{\text{optimal}}]/\text{score}_{\text{optimal}})$ so that optimal is 0%.

Feature Selection Type	Learning Method	Learning Method Output	NYC-MSP Mon-Fri 0,1,2 stops	NYC-MSP M-F nonstop	NYC-MSP Tu-Th 0,1,2 stops	NYC-MSP Tu-Th nonstop
			(mean cost (in $), percentage above optimal (PAO, in %))			
Deterministic Feature Selection	Earliest Purchase	Buy/Wait	(317, 18.2)	(414, 21.4)	(309, 17.5)	(374, 24.2)
	Optimal Cost	Buy/Wait	(268, 0.0)	(341, 0.0)	(263, 0.0)	(301, 0.0)
	Bing Travel	Buy/Wait	(308, 14.9)	N/A	(306, 16.3)	N/A
	PLS w/ Most Recent Obs.	Regression	(314, 17.1)	(384, 12.6)	(294, 11.8)	(354, 17.6)
	PLS w/ Full Lag Scheme	Regression	(300, 11.9)	(398, 16.7)	(316, 20.1)	(345, 14.6)
Off-the-shelf Methods	PLS w/ CFS	Regression	(313, 16.8)	(413, 21.1)	(308, 17.1)	(371, 23.2)
	PLS w/ BFS	Regression	(317, 18.2)	(416, 22.0)	(310, 17.8)	(369, 22.6)
Developer-Guided Feature Selection	REPTree in Classification	Buy/Wait	(288, 7.4)	(388, 13.8)	(289, 9.9)	(382, 26.9)
	REPTree in Regression	Regression	(284, 5.9)	(375, 10.0)	(280, 6.5)	(334, 11.0)
	Ridge Regression	Regression	(293, 9.3)	(383, 12.3)	(316, 20.1)	(372, 23.6)
	nu-SVR	Regression	(295, 10.1)	(396, 16.1)	(289, 9.9)	(338, 12.3)
	PLS Regression	Regression	(280, 4.5)	(365, 7.0)	(276, 4.9)	(330, 9.6)

to the test set achieves a PAO 1–3% closer to optimal than earliest purchase. Since Bing Travel does not separately predict for non-stop flights, there are no results for the non-stop category.

For deterministic feature selection approaches, there are several benchmarks. The *Most Recent Observation* with PLS regression contains only the most recent value from each raw feature (i.e. no lagged variables). This method cannot leverage trends but can observe competitive relationships. The *Full Lag Scheme* with PLS regression contains all lags $\{0, -1, \ldots, -7\}$ from every feature. The method can predict trends and can leverage competitive relationships, but prediction performance may suffer due to the large number of features.

Off-the-Shelf Methods

PLS w/ CFS is Correlation-based Feature Selection, a filter based automated feature selection method, coupled to PLS regression after the feature selection step. Its performance is worse than the performance of PLS regression alone (i.e. with no feature selection). The correlation between the target variable and individual features may not provide a predictive feature set in aggregate.

PLS w/ BFS is best first search, a wrapper-based automated feature selection method utilizing greedy search, coupled to PLS regression for each feature set evaluation. The results are worse than earliest purchase. The approach may be poor due to the large search space caused by many raw features.

Developer-Guided Feature Selection Methods

In the experiments for this application domain, Developer-Guided Feature Selection is coupled to five different learning algorithms using the feature classes and constraints given earlier. These experiments enumerate all valid feature sets given the constraints. The scores are for the best model found on the calibration set which is then applied to the test set. The decision tree algorithm, *REPTree*, is used in both classification mode (to produce buy/wait signals directly) and regression mode (to build price predictions which are fed to the decision threshold function). *REPTree in regression* performs well. It implicitly performs a feature selection process when building the tree so it may be able to prevent overfitting due to a large number of collinear features.

Ridge regression is a regularized linear regression algorithm used in multivariate domains with many (possibly collinear) variables. It may perform poorly due to overfitting because the number of observations (≈ 100) is insufficient relative to the number of features (≈ 1000). This makes overfitting more likely.

nu-SVR, a support vector machine method (with linear kernel), performs almost as well as PLS regression for this domain, but it is much more computationally expensive due to the large number of features for some lag scheme configurations.

PLS regression performs the best in this application within $\sim 6.5\%$ of optimal cost on average from the targets in Table 5.6. The method implicitly does a dimensionality reduction on the training set concept during calibration, so it is more robust to overfitting due to the small number of training samples and large number of features.

Table 5.7 shows that the optimal policy provides on average a 11.0% savings over earliest purchase. We denote this percentage as the *savings margin*. Our method of a lag scheme search coupled with PLS Regression and a decision threshold achieves consistently closer to the optimal action sequence than any of the other methods compared. The PAO of $\sim 6.5\%$ achieved by PLS

(in Table 5.6) represents a savings of 8% over the earliest purchase strategy for the NYC→MSP route.

5.5.2 Bing Travel Performance Comparison

Bing's Travel provides a prediction about whether or not the lowest cost ticket from any airline will be lower over the next 7 days. Even though Bing's Travel has different and broader objectives than our study, it is surprising that it does not achieve a greater savings margin on the Any Airline target. We believe that this is due to its more risk averse approach that makes it significantly more likely than our method to advise immediate purchase.

This assertion can be validated by looking at the distribution of buy and wait signals computed for each day by the various policy generators: in the NYC→MSP M-F route, the optimal policy has only a 15% proportion of buy signals. It is noteworthy that the best models constructed with our method emits a similar proportion of wait signals: in the NYC→MSP M-F route, the model with the lowest average cost ($280) only emits a buy signal on 34% of the days. Bing's model has a much higher proportion of buy signals: in the same route, the Bing model emits a buy signal 83% of the days. The results are similar for all routes and dates in our dataset: Bing emits buy signals for at least 70% of the days.

5.5.3 Multi-route Comparison

To show that the proposed method is generalizable to other routes (including international routes), we provide performance statistics on 7 routes in Table 5.7. The proposed method achieves an average of 69% of the optimal savings which represents an average cost savings of 7.25% when compared to the earliest purchase strategy. Given the high cost of airline tickets, this represents a significant savings. For the purposes of comparison with existing approaches, we provide results of two decision theoretic methods from [Etzioni et al., 2003]: Ripper and LR (an MA(1) time series model in the notation of [Box et al., 2013]). Those models use a smaller number of features compared to our model and do not leverage the competitive relationships between airlines when making predictions. We believe predictions are improved by considering price competition between airlines.

5.5.4 Specific Preference Models

So far we have shown how our model predicts the expected minimum price of all available flights on a specific route and departure date. We now show how our model can also predict prices of flights with specific desirable properties, such as flights from a specific airline, non-stop only flights, or multi-segment flights. Buyers are likely to have preferences about airline tickets

Table 5.7: Percentage above optimal cost (PAO, as %) for various decision theoretic approaches tested on the 7 (domestic and international) routes shown (using the airport codes) for a 5-day round trip both for Monday and Thursday departures. (Note: "PAO" is computed as a proportion: $(\text{cost} - \text{cost}_{\text{optimal}}) \div (\text{cost}_{\text{optimal}})$.) Cities are: HOU=Houston, TX, USA ; MSP=Minneapolis, MN, USA, NYC = New York, USA; CDG = Paris, France; CHI = Chicago, USA; HKG = Hong Kong, China; SEA = Seattle, WA, USA; IAD = Washington, DC, USA. Savings Margin computed as % of earliest purchase cost and represents the savings a strategic consumer may experience using the model. The "$*$" denotes an international route.

	Model					
	Optimal	Our Model	Linear (LR)	Ripper	Earliest Purch.	Latest Purch.
Method Origin:	baseline	this paper	[Etzioni et al., 2003]		baseline	baseline
HOU → NYC	0.0	4.3	13.9	19.8	14.8	53.9
MSP → NYC	0.0	3.9	14.5	21.7	14.3	48.8
NYC → CDG$*$	0.0	5.5	14.5	26.1	16.6	59.0
NYC → CHI	0.0	6.8	15.7	48.1	28.4	108.5
NYC → HKG$*$	0.0	6.4	13.3	36.0	17.3	39.0
NYC → MSP	0.0	5.2	14.0	33.0	18.5	58.1
SEA → IAD	0.0	4.5	14.2	18.8	12.8	43.3
Mean PAO	0.0	5.3	14.3	28.9	17.4	58.7
Savings Margin (%)	11.0	7.25	0.514	-10.4	0.0	-32.3

beyond price, such as loyalty to a specific airline or the desire for overall minimum travel time. By comparing models with different target properties, buyers can determine the likely cost of their preferences.

A model for a specific preference (e.g. non-stop only flights, flights with take-off time before 11 am, or flights from a specific airline only) can be generated by computing the correct price prediction target variable to train the regression model. The regression model's target price for each observation day will depend on the exact flight preferences. For example, the future minimum price of any-airline and any number of stops (ALL-min-A) for the departure date will always be equal to or less than the future minimum price for a specific airline and non-stop only flights (e.g. DL-min-0). In the terminology of machine learning, this corresponds to computing the label for each row in the dataset. The target label values will depend on the concept being learned. And these values are computed from the subset of the airline itineraries that match the preference specified.

The effect of more specific preferences can be observed in the feature set configurations that are generated for each preference. In Table 5.8, the legacy airlines (DL and CO) offering multiple types including non-stop and multi-stop flights may have different lag schemes based on the preference. The general pattern observed is that more specific preferences (such as non-stop only flights) require more information in terms of more specific feature classes and more time lags than less specific preferences (such as any flight from any airline). Also, more desirable

Table 5.8: Optimal lag schemes for specific preferences of airline and number of stops. Experiments are for 5-day round trips departing on Thursdays. The statistics are as follows: "MC" refers to average model cost, "EP" refers to average earliest purchase cost, and "PAO" is the mean price of the model above the optimal policy price (expressed as %, closer to zero is better).

Legacy Airlines

Continental Airlines (CO) — Non-stop

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A	•	•	•	•	•	•	•	•
ALL-S	•	•	•	•	•	•		
EACH-A		•	•	•	•	•		
EACH-S				•				

(MC: $452.7, EP: $483.6, PAO: 7.0%)

Continental Airlines (CO) — 0, 1, or 2-stops

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A	•	•	•	•	•	•		
ALL-S	•	•	•	•	•	•		
EACH-A								
EACH-S								

(MC: $344.8, EP: $386.3, PAO: 4.4%)

Continental Airlines (CO) — 1-stop

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A		•	•					
ALL-S		•	•					
EACH-A		•						
EACH-S								

(MC: $422.7, EP: $456.1, PAO: 3.3%)

Delta Airlines (DL) — Non-stop

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A			•	•	•	•	•	•
ALL-S			•	•	•	•	•	•
EACH-A			•					
EACH-S			•					

(MC: $389.5, EP: $411.6, PAO: 7.1%)

Delta Airlines (DL) — 0, 1, or 2-stops

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A	•	•	•	•	•	•	•	•
ALL-S	•	•	•					
EACH-A		•						
EACH-S		•						

(MC: $366.8, EP: $393.8, PAO: 6.7%)

Low-Cost Airlines (LCAs)

Frontier (F9) o — 0, 1, or 2-stops

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A			•	•	•	•	•	•
ALL-S				•	•	•	•	•
EACH-A						•	•	
EACH-S						•	•	

(MC: $319.0, EP: $357.2, PAO: 3.5%)

AirTran (FL) — 0, 1, or 2-stops

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A	•	•	•	•	•	•	•	•
ALL-S			•	•	•	•	•	•
EACH-A								•
EACH-S								

(MC: $315.0, EP: $335.7, PAO: 3.2%)

Sun Country (SY) — Non-stop

Class	Lagged Offsets							
	0	-1	-2	-3	-4	-5	-6	-7
DET	•							
ALL-A		•	•	•	•	•	•	
ALL-S		•						
EACH-A		•						
EACH-S								

(MC: $352.6, EP: $399.1, PAO: 1.3%)

flights such as non-stop only flights will tend to be more expensive than multi-stop ones. This is also observed in the lag scheme plots: our model is able to obtain an average price of $452 for Continental (CO) non-stop flights, but can obtain an average price of $345 for flights possibly with multiple stops.

For the LCAs, the lag schemes suggest that there are delays in each airline's response to changes in prices. For example, the LCA models use little (AirTran) or no (Frontier and Sun Country) information from the current day (lag offset 0) and previous day (lag offset -1) for prediction. An alternative explanation for no current day information in the model may be in the airline's role as a price leader. If an airline is setting prices aggressively, the airline will not make rapid changes in response to the actions of other companies in the market.

Table 5.8 shows that some LCA models (Sun Country and AirTran) do not use information at the Each-Stops level. This suggests that these airlines do not use detailed information from other airlines to determine their responses. The prices for these airlines generally are set equal to or just below the prices of the legacy airlines, so detailed information about the legacy airlines' pricing for non-stop and multi-stop flights may not be needed for prediction. Also, because the pricing relationships for the LCAs is less sophisticated than the relationships of the legacy airlines, the prediction models should be more effective for predicting LCA pricing than for legacy airline pricing. This can be observed in the percentage above optimal (PAO) values observed for LCAs compared to legacy airlines.

Using these statistics, it is also possible to reason about the relative costs of various preferences: for example, what is the expected price difference of a non-stop Delta flight and a flight on any available airline? Such information could help customers to quantify the expected costs of their preferences.

5.5.5 Feature Class Hierarchy Sensitivity Analysis

The principal benefit of Developer-Guided Feature Selection is from a reduction in the feature selection search space through the use of the practitioner-provided domain knowledge. This section considers how sensitive the results are to the constraint set choice. Experiments in Table 5.9 examine the performance of alternative constraint sets. The original constraint set presented in Figure 5.6 is labeled as "Constraint Set A" and these results are consistent with results in Table 5.7. We modify the constraint set by swapping the location of All-Stops and Each-Aggregate; this is labeled as "Constraint Set B." Another modification increases the range of time lagged variables from $[0, -7]$ to $[0, -10]$ time units, labeled as "Constraint Set C." This greatly increases the configuration search space. "Constraint Set D" involves a search with no constraints in the feature class hierarchy: all nodes are at the same level in the tree and there are not subset constraint relationships. The no constraints search space is considerably larger

and involved 1,414,562 possible configurations.

Table 5.9: Developer-Guided Feature Selection constraint set sensitivity analysis.

Learning Method	Feature Class Hierarchy	NYC-MSP Mon-Fri	NYC-MSP M-F nonstop	NYC-MSP Tu-Th	NYC-MSP Tu-Th nonstop
		(mean cost (in $), percent above optimal cost (PAO, in %))			
PLS Regression	All Lags	(300, 11.9)	(398, 16.7)	(316, 20.2)	(345, 14.6)
	Most Recent	(314, 17.2)	(384, 12.6)	(294, 11.8)	(354, 17.6)
	Constraint Set A	(280, 4.5)	(365, 7.0)	(276, 4.9)	(330, 9.6)
	Constraint Set B	(291, 8.6)	(393, 15.2)	(279, 6.1)	(330, 9.6)
	Constraint Set C	(297, 10.8)	(401, 17.6)	(280, 6.5)	(353, 17.3)
	Constraint Set D	(292, 9.0)	(407, 19.4)	(280, 6.5)	(348, 15.6)

These experiments find that a different constraint set does not dramatically change the test set scores (A vs B). Increasing the range of available lags beyond a 7 day pattern does not improve results (C). We conjecture that all constraint sets (A, B, C, and D) improve upon the performance of the no feature selection version because the feature selection process reduces the number of features used in model calibration. This is especially beneficial for domains where the number of observations (n) is small relative to the number of features (k): $n < k$. The exact choice of constraints is not critical. Of primary importance is the need to reduce the feature sets evaluated.

Regarding the performance differences between the constraint sets, we conjecture that the performance degradation observed by constraint set D is due to oversearching, the phenomenon of a search process exploiting transient patterns in the dataset that do not generalize well. As the number of configurations evaluated increases, so does the likelihood of oversearching [Quinlan and Cameron-Jones, 1995]. Constraint sets A, B, and C explore different (and much smaller) subsets of the search space than D. Constraint set A is the one most congruent with our understanding of the variable relationships in this domain, and we find its performance to be similar to or superior to other configurations.

5.5.6 Prediction Accuracy of Purchase Timing

Another way to analyze and compare the action signals from the methods presented is to consider the temporal accuracy of the purchase timing for each simulated passenger. Because the exact sequence of future prices is known for each simulated passenger, the exact number of days to wait to achieve the optimal (lowest) purchase cost is known. We can compare the number of days the algorithm emitted a wait signal against the optimal number of days to wait. This comparison is shown visually for the test set observations in Figure 5.7. The methods that emit markers closer to the origin are best because they are emitting a sequence of buy-wait signals

closes to the optimal policy. In practice, this is difficult to achieve.

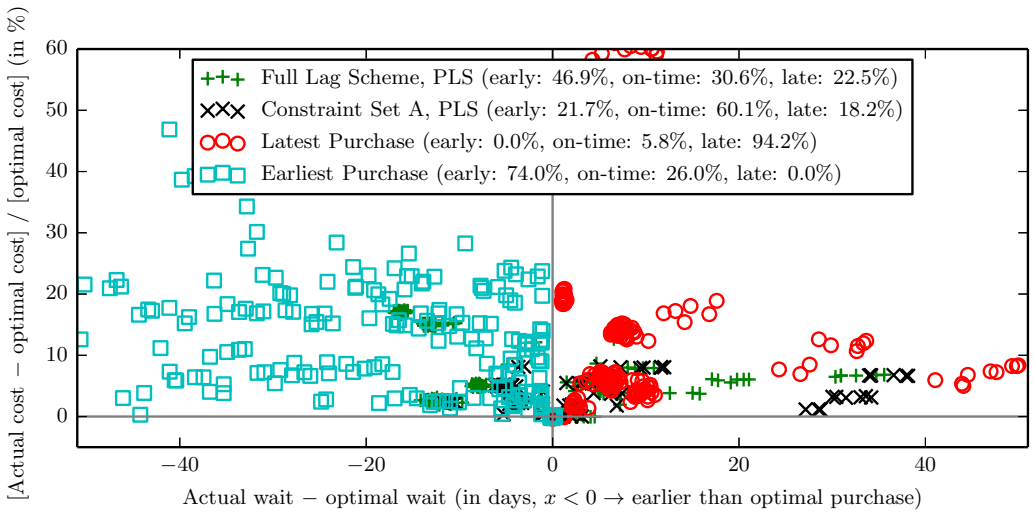


Figure 5.7: Temporal accuracy of purchasing signals for 4 purchase policy generators. Each marker corresponds to an individual purchase. Points with $x < 0$ indicate purchases that were before the optimal purchase time (categorized as "early") while points with $x > 0$ indicate purchase signals that waited more than the optimal number of days (categorized as "late"). The y-axis value indicates how much higher the purchase cost was than the optimal cost (denoted as "PAO", percentage above optimal). Gaussian noise is added to help visualize overlapping markers.

We show the results of earliest purchase (always emit a buy signal) and the results of latest purchase (emit a wait signal until the last possible day) as benchmarks. As a simple check, we can see using the earliest purchase results that the best price is achieved using an always-buy signal only 26.0% of the time. Also, waiting until the last possible time to buy (an always-wait signal) only achieves the lowest possible cost 5.8% of the time. Both of these approaches increase the overall cost above the optimal policy by (∼15%) as shown in Table 5.10.

The two complex methods we show on the graph are "Full Lag Scheme, PLS" which is the results using our decision policy framework but without any feature selection search. As this method uses all the variables and time lags available we expect that the price predictions may be more sensitive to movements of some variables. In the time accuracy plot this method is much more likely to buy before the optimal purchase timing (46.9% of the purchases are before the optimal purchase timing). The purchase timing results of the best method based on our experiments is shown as "Constraint Set A, PLS". It is able to achieve the optimal purchase timing on 60.1% of all simulated purchases. It reduces the proportion of both late and early purchase signals. Also, it is possible to see from the markers in Figure 5.7, even for late and early signals, the prices experienced by the best method are closer to the optimal price.

Table 5.10: Statistics for different purchase policy generators.

Method:	Full Lag Scheme			Constraint Set A			Latest Purchase			Earliest Purchase		
Lateness Bin	early	on-time	late	early	on-time	late	early	on-time	late	early	on-time	late
Bin Percentage (%)	46.9	30.6	22.5	21.7	60.1	18.2	0.0	5.8	94.2	74.0	26.0	0.0
Mean Lateness (in days)	-10.9	0.0	9.3	-5.0	0.0	13.6	—	0.0	10.1	-17.7	0.0	—
PAO (% above optimal)	9.2	0.0	4.3	4.1	0.0	4.0	—	0.0	15.8	13.6	0.0	—

5.5.7 Market Competition

Varying market competition is one potential explanation for the differences in performance across routes. We examine this in more detail by computing market competition for each route and plotting it with the savings margin. To measure market competitiveness, we use the Herfindahl-Hirschman Index (HHI), an economics-based measure of market consolidation [Rhoades, 1993]. The index is computed on the market share percentages of all participants and ranges from almost zero to 10,000 (perfect monopoly):

$$\text{HHI } = \sum_i ({m_i}^2), \text{ where } m_i \text{ is the percentage market share of participant } i \tag{5.6}$$

Markets having an HHI value greater than 2,500 are considered to be very noncompetitive. Market share information is computed for US domestic airline routes from a dataset containing a sample of all domestic airline tickets issued in quarter 3 of 2011 [U.S. Department of Transportation, 2012].

Table 5.11: Effect of market competition on model performance. Market competition statistics can be generated for US domestic routes using data from U.S. Department of Transportation [2012]. Routes are ordered by decreasing competition (increasing HHI value).

Airline Route	No. of Significant Competitors	Mean Passengers per Day	Competition (HHI)	Earliest Purchase Std. Dev. (as %)	Our Model Savings Margin (as %)
NYC-CHI	9	6686	1861	6.79	12.834
MSP-NYC	6	1597	3228	5.65	8.958
NYC-MSP	7	1624	3296	5.60	7.091
HOU-NYC	5	2305	4484	5.53	6.615
SEA-IAD	2	385	7790	4.55	7.056

Some routes are naturally more competitive than others due to the number of airlines that have market power and price aggressively to affect demand. Airlines that actively serve a route are ones that handle a significant volume of the route's traffic. We define a *significant competitor*

on a route as an airline serving more than 1% of all passengers. By comparing the results of our proposed prediction model across routes, it is possible to see the effects of competition on the model savings. Table 5.11 reveals that for the most competitive routes (e.g. NYC-CHI, MSP-NYC) the savings margin experienced by the model is larger at around 10%. For routes that are are less competitive and have a larger HHI value (e.g. SEA-IAD, HOU-NYC), the savings margin is reduced. Competition is also somewhat correlated with market size (as measured by mean passengers per day), but our results find that competition is a better predictor.

Another correlated measure is seen by looking at the variance of the prices from the earliest purchase strategy as measured using standard deviation in Table 5.11. The lower variance of daily minimum prices shown in the NYC-CHI route is likely due to the large number of competitive carriers along the route and the large number of passengers. In contrast, the SEA-IAD route has fewer "significant competitors", so individual airlines can assert greater pricing power. An analysis in [Vowles, 2000] confirms a similar behavior of dominant competitors using a regression analysis of fares on US domestic routes.

5.6 Conclusions and Future Work

We have presented a method for predicting airline ticket prices. The method uses historical data from which relevant features are extracted to build a predictive model of prices for all the airlines serving an origin-destination pair of airports on a specific departure date. Our results show that, given publicly-observable historical data, airline ticket prices can be predicted. While buying at the earliest opportunity is the most obvious purchase policy, we show that is not the best policy in most cases. The long lead-time price may not be the lowest price available for that flight. There is also an opportunity cost associated with early commitment: a customer risks being locked into a specific schedule that may need to be changed (for a fee).

Because there is sufficient structured price volatility on many airline routes, there are significant opportunities for savings. To our knowledge, our results represent the state-of-the-art in airline ticket price prediction using consumer-accessible data. We believe that there is a significant market for these kinds of models in the hands of consumers not only to reduce their travel costs, but also to determine the range of expected prices for an itinerary, and to quantify the cost of airline and routing preferences.

The main novelty of our approach lies in our Developer-Guided Feature Selection technique, which captures temporal dependencies in the data via time-delayed features, and which reduces the number of features by using a class hierarchy among the domain features and pruning them based on in-situ performance.

Developer-guided feature selection has wide applicability to other multivariate domains where

basic domain knowledge is common but not utilized and where there are significant intra-variable and inter-variable temporal relationships. Building the feature class hierarchy requires only basic domain knowledge to be successful; greater expertise in hierarchy construction could improve efficiency even more. Feature selection contributes to prevent overfitting (evident from the poor performance of off-the-shelf feature selection approaches) which can occur when there are many features and few training instances. In addition, it enables the discovery of meaningful relationships among features and facilitates domain understanding. By examining the relative performance of candidate lag schemes, domain knowledge can be extracted: the significance of individual features can be determined by observing their presence (or absence) in the calibrated lag scheme and feature set.

Dynamic pricing could be beneficial in other types of industries. For instance, industries that store inventory traditionally have concentrated their efforts on tracking inventories to reduce inventory size, because of the high cost of changing prices [Elmaghraby and Keskinocak, 2003]. However, the availability of demand data, the ability to inexpensively change prices, and the use of decision support tools can change the situation and bring dynamic pricing to industries that currently rely on inventory control (see [Groves and Gini, 2013a] for an example in simulated supply-chain management.)

From the consumer perspective, optimizing purchase timing is beneficial in other markets as well. While consumer goods markets are clearly different from the airline ticket market, there are similar systematic behaviors which could be leveraged through prediction [Agrawal et al., 2011]. In auction markets, such as the eBay online marketplace [Ebay Inc, 2012], properties of auctions, such as ending hour, seller feedback rating, etc., have been shown to affect the final auction prices [Bajari and Hortacsu, 2003; Lucking-Reiley et al., 2007]. Such information has been successfully applied in auction arbitrage [Raykhel and Ventura, 2009].

Chapter 6

Applied Domain: Stream/River Flow Prediction

Stream flow prediction is an important real-world application domain for time-series prediction. We present river flow forecasting as a proof-of-concept domain for the Developer-Guided Feature Selection (DGFS) framework. DGFS can improve prediction model quality over a straightforward application of standard machine learning models and feature selection methods. This is not to assert that DGFS is the best approach for the domain, simply that it represents an improvement over general approaches.

As this domain has significant temporal and spatial aspects, it is a natural fit for the DGFS framework. The domain can be represented as a spatial data network in which the observation sites are spatially related but have fixed locations and relationships [Shekhar and Chawla, 2003]. We propose to apply DGFS in order to build prediction models for short-term (1 to 120 hours ahead) flow forecasting at a specific measurement site. We use several types of data streams in the raw feature vector including:

- upstream/downstream site readings and
- predictions of future precipitation at many sites in the river basin.

These data streams may have different data resolutions, but the framework can handle these different observation resolutions concurrently. We see this as a possible advantage of our approach. The following is a discussion of the architecture applicable to stream flow prediction as well as some challenges.

6.1 Domain Description

The river network targeted for this analysis is the Mississippi River which runs north to south in the central United States. It is the longest river in North America and has a river basin land area of 1,250,000 square miles (approximately 40% of the land area of the contiguous US). There are many year-round measurement sites along the network with hourly readings. A small subset of 6 continuous measurement sites is used for this study in the upper portion of the network. These data are presented at one-hour resolution. This experiment proposes to build a regression model that combines flow readings (at one-hour resolution) and precipitation measurements (at six-hour resolution) to accurately predict future flows forward in time at a specific gauge site.

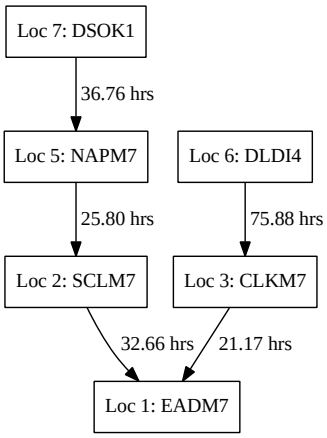


Figure 6.1: Physical relationships between observation sites for downstream prediction target site EADM7 at St. Louis, MO. The period of maximum correlation (in hours) between adjacent measurement sites (computed experimentally from the data) is labeled for each edge. Each directed edge represents a *flows into* relationship (i.e. the upstream site flows into the downstream site).

A sample network in the stream flow domain is shown as a directed graph in Figure 6.1. When constructing a prediction model for site "EADM7"[1], there is a network of upstream gauge sites along the river and its tributaries that may provide predictive information about the target. The directed edges in the figure denote the *flows into* relationships between adjacent sites along the network. In leveraging the spatial aspect of the problem, we assume that downstream sites will have no affect on upstream observations, consistent with a conventional understanding of the domain as water flows only downstream. It is expected that there are temporally lagged relationships between adjacent gauge sites: an impulse of flow at site A will appear at a downstream site B after a fixed time duration of x hours, where $x \in [0, w]$ and w is the maximum

[1]Mississippi River at St. Louis, MO, USA

temporal lag and is set during initial model construction. These temporal lag relationships were empirically measured from the observation data and are provided in Figure 6.1 for comparison.[2]

Two types of experiments are made for this domain:

- first, only river flow observations will used in the raw feature set;
- second, precipitation data are added to the feature set (using an additional set of feature classes) in an attempt to anticipate changes in flow due to precipitation events.

For this domain, each feature class consists of observations from a single measurement site. The feature class hierarchy is constructed from the physical relationship tree (Figure 6.1). Constraint relationships between sites will also be incorporated in the feature selection search: valid time lags assigned to each feature class are constrained to values that are equal to or greater than the time offset for any downstream feature class (reachable by forward traversals of one or more *flows into* links). Given the set of n feature classes ($F = (f^{(1)}, f^{(2)}, \ldots, f^{(n)})$) in the domain and a set of constraints (Table 6.1), constraints are enforced using the logical formula in Equation 6.1. δ_i is the time lag assigned for feature class i.

$$\forall_{f^{(i)} \in F} \forall_{f^{(j)} \in F} [(i,j) \in \text{CL}] \implies [\delta_i \geq \delta_j] \tag{6.1}$$

CL refers to the set of feature class constraints for the domain. $\delta_i \leq \delta_j$ means the time offset of feature class i must be greater than (later in time) or equal to the time offset of feature class j. For example, the feature class containing the observation at site #5 "NAPM7" will have an offset less than or equal to the lag offset for site #2 "SCLM7": $\delta_2 \geq \delta_5$.

Feature Class Constraints (CL)
$(f^{(2)}, f^{(1)})$
$(f^{(3)}, f^{(1)})$
$(f^{(5)}, f^{(1)})$
$(f^{(6)}, f^{(1)})$
$(f^{(7)}, f^{(1)})$
$(f^{(5)}, f^{(2)})$
$(f^{(7)}, f^{(2)})$
$(f^{(6)}, f^{(3)})$
$(f^{(7)}, f^{(5)})$

(a) flow information only experiments

Feature Class Constraints (CL)	
$(f^{(2)}, f^{(1)})$	$(f^{(21)}, f^{(11)})$
$(f^{(3)}, f^{(1)})$	$(f^{(31)}, f^{(11)})$
$(f^{(5)}, f^{(1)})$	$(f^{(51)}, f^{(11)})$
$(f^{(6)}, f^{(1)})$	$(f^{(61)}, f^{(11)})$
$(f^{(7)}, f^{(1)})$	$(f^{(71)}, f^{(11)})$
$(f^{(5)}, f^{(2)})$	$(f^{(51)}, f^{(21)})$
$(f^{(7)}, f^{(2)})$	$(f^{(71)}, f^{(21)})$
$(f^{(6)}, f^{(3)})$	$(f^{(61)}, f^{(31)})$
$(f^{(7)}, f^{(5)})$	$(f^{(71)}, f^{(51)})$

(b) flow information and precipitation experiments

Table 6.1: Constraints between feature classes.

[2]Each time lag was determined as the offset (in hours) which achieves the maximal positive correlation between the two measurement time series.

6.1.1 Source Data

The dataset for these predictions is publicly available data from the US National Oceanographic and Atmospheric Administration (NOAA) for hourly flow observations at each gauge site. This represents a contiguous hourly observation period of 5959 samples from Jan 15, 2013 to September 15, 2013. Figure 6.2 shows the model testing configuration used. As the dataset is time ordered, the order must be preserved to avoid leaking data between training and testing. To clarify, the training set is the first 33% of the observation sequence, the prediction model optimization is performed through repeated evaluations on the validation set (the middle 33%), and the reported scores are the predictions obtained for the test set (the last 33%) of the observation sequence. This experimental protocol was chosen so that all model testing periods (train, validate, and test) contain significant precipitation events. This can be visually verified by inspecting the flow and cumulative precipitation data in Figure 6.3.

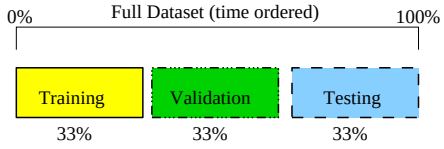


Figure 6.2: Training/validation/testing set split used for the river flow domain.

Several models for different prediction horizons (number of hours ahead for which predictions are made) were constructed for this domain. Short-term predictions provided by the NOAA for the EADM7 gauge site are also used for a comparison benchmark.[3]

6.1.2 Periodicity

The time series of many natural phenomena contain consistent periodic patterns. The time series literature leverages this aspect through the autoregression ("AR(*)") model type for the prediction of periodic patterns. This aspect also motivates the use of time lagged observations in prediction models based on machine learning methods. This can be explained by the daily periodicity present in the data. In this domain, there exists daily periodicity in the observations at some sites which is possibly due to fluctuations in the usage of hydro-power generating capacity at upstream sites [Kern et al., 2011]. It is natural to observe the previous day's (offset exactly 24 hours earlier from the target prediction time) value as more predictive than the most recent available observation.

[3]The precise methodology used to compute the short term predictions is not published by the agency other than to describe it as an "ensemble model." We believe these models are tuned using expert analysis specific to each site. These predictions are only available for a subset of the measurement sites.

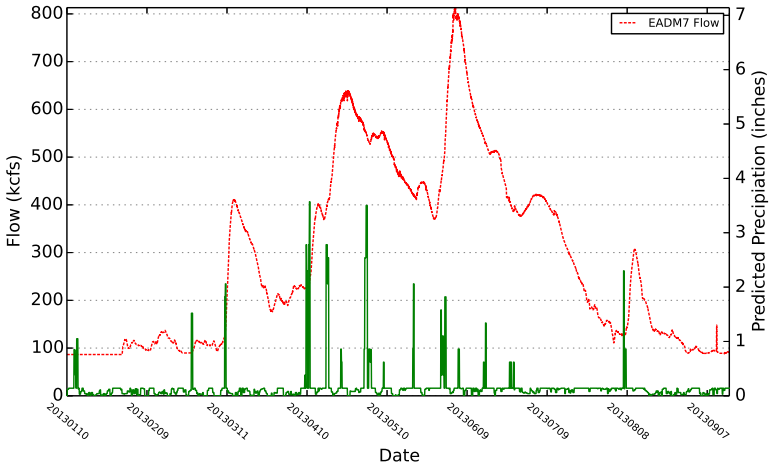


Figure 6.3: Hourly flow observations for gauge site EADM7 (left y-axis) and sum of predicted precipitation of entire river basin (every 6 hours on right y-axis) for test set time period.

This phenomenon can be observed in the data. Autocorrelation at a 24 hour cycle may not be initially obvious on visual inspection of the flow time series (Figure 6.3). However deeper analysis of the data shows evidence of daily auto-correlations. A strong daily cycle can be observed by statistically examining the first derivative of flow measurements. Figure 6.4 shows evidence of high positive correlation of the first derivative of the flow for gauge site EADM7 around time offsets of 24 hour intervals: 24 hours, 48 hours, and 72 hours. Also, anti-correlation is observed for 12 hours and 36 hours. This observation may explain why time lagged observations far from the peak observed correlations are included in the lag scheme. A time offset that is a multiple of 24 hours away from best time lag may still be quite informative for prediction.

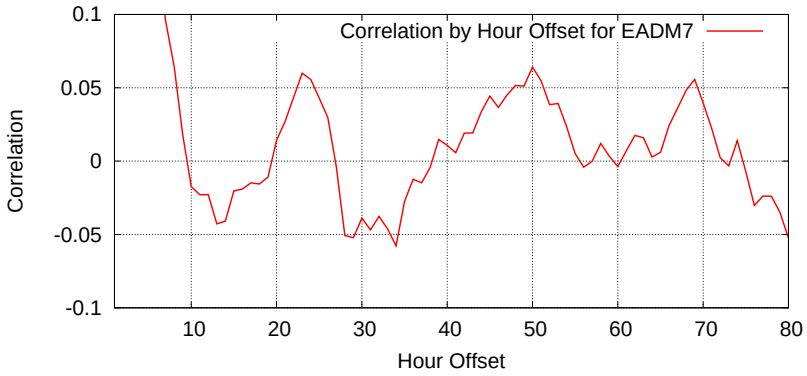


Figure 6.4: Correlation coefficients computed by comparing the rate of change of the EADM7 flow time series against itself at specific hourly offsets. The series measures autocorrelation at increasing hourly intervals.

6.2 Experimentation

To maximize the value of this investigation, the exposition here describes the empirical performance observed for several possible alternative approaches for prediction in this domain. We hope that this approach will inform the reader 1) about the performance characteristics of several choices made when building models with the DGFS framework and 2) that common approaches advocated in the literature can have empirically poor performance on real data.

A sequence of several prediction models are made for this domain with increasing complexity in terms of the model, algorithm, and dataset used. The exposition is sequenced as follows:

1. univariate methods (single target, time series prediction),
2. multivariate methods (no temporal aspect),
3. multivariate methods with temporal aspect,
4. data driven feature selection with temporal aspect,
5. Developer-Guided Feature Selection (DGFS), and
6. impact of errors in constraints on DGFS.

The performance of each of these model types is evaluated in turn. In applying DGFS to the domain of river flow prediction, we use the existing approaches both from the domain and from the machine learning literature as performance baselines. Also, the US NOAA computes short-term flow predictions for major measurement sites on the river network and we use these as a performance baseline. The details of this method are not made public, so we have no expectations that the performance of our model would improve upon this model developed by domain experts. The purpose of this chapter is to compare results developed using our feature selection methodology against approaches advocated in the literature for general-purpose

application to real-world regression domains.

The experiments in this section are for future flow prediction for a single gauge site "EADM7" in the Mississippi river network. This site has NOAA predictions for up to 200 hours into the future which we use for comparison. Flow data for several major upstream measurement sites is available for the data collection period. We compute a flow prediction for 12, 24, 48, 72, 96, and 120 hours from the observation time for every hour ($\sim$ 2000 hours) in the test dataset.

6.2.1 Univariate Methods (single target, time series prediction)

		Target Offset					
Method	**Algorithm**	12	24	48	72	96	120
lastvalue	PLS	12.92	24.06	44.81	63.19	79.09	92.70
lastvalue	Ridge	12.92	24.06	44.81	63.19	79.09	92.70
lastvalue	nuSVR	12.64	23.48	43.46	61.11	76.43	89.10
mostrecent	ZeroR	201.10	200.83	200.35	199.97	199.66	199.44
uts8h5d	PLS	9.77	18.32	36.63	55.16	72.26	87.02
uts8h5d	Ridge	9.63	18.18	36.48	54.99	72.13	86.89
uts8h5d	nuSVR	11.89	20.87	38.32	55.62	71.27	84.68

Table 6.2: Test set flow (kcfs) RMSE for univariate methods.

Past observations of the target variable may be predictive of future values. The approach of univariate time series models (of [Box et al., 2013]) is applied here. A very primitive prediction baseline, "ZeroR", uses the mean value from the target feature in the training set as the prediction for all observations in the validation set. This method has roughly equivalent performance for all time horizons ("offsets"), as expected. An alternative, "lastvalue", uses the last observed value at the measurement site as the prediction. This is often called a *persistence model* and is another essential baseline for comparison. For 48 hours ahead predictions, this is equivalent to using the value observed 48 hours earlier as the prediction for the current observation. More complex prediction methods should reliably exceed the accuracy of these two approaches to be considered useful for prediction. Including these two methods in our analysis provides a baseline for measuring the effectiveness of prediction in the domain.

Table 6.2 shows the "ZeroR" baseline finds mean errors of approximately 200 kcfs (thousands of cubic feet per second), the largest error of any method. Also, the error is consistent over the range of prediction horizons.[4]

[4]The variation in results is due to small differences in the datasets used for each prediction horizon. For example, the 120 hour ahead predictions will be over a slightly different dataset than the 24 hour ahead predictions even though both datasets have the same number of observations. The exact sequences for each target will differ due to truncation of the target variable's data at the beginning or end of the dataset due to the time horizon

In addition, the performance of purely univariate methods is observed by the experiments in Table 6.2. A univariate time series model can be constructed by using a set of time-delayed (lagged) variables taken from the prediction target. A set of common machine learning models is applied to this dataset including Ridge Regression ([Hoerl and Kennard, 2000]), Partial Least Squares Regression (PLS, [Wold et al., 1983]), and Support Vector Regression (SVR, with linear kernel). These univariate regression methods are evaluated using lagged observations for the last 5 days at 8 hour intervals ("utm8h5d"). The univariate time series methods have improved performance over the "last value" methods, which suggests that these simple univariate time series predictions are beneficial in this domain.

The univariate time series models can have the input feature set visualized using the lag scheme methodology presented in this thesis. Table 6.3 shows the lag scheme which generates the univariate time series "utm8h5d". The markers in the table indicate the class and offset pairs that are included in the augmented feature set provided to the learning algorithm. In this configuration, there are 15 instances of feature class "1" (EADM7) in the augmented feature set.

	Lagged Offsets (hours)														
Class	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80	-88	-96	-104	-112
1	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
2															
3															
5															
6															
7															

Table 6.3: The "utm8h5d" lag scheme which generates the univariate time series. A bullet ("•") indicates the presence of a feature class-time lag pair in the augmented feature set used in learning.

6.2.2 Multivariate Methods (with no temporal aspect)

Another approach to prediction in this domain is to use a multivariate prediction model to perform regression over many dissimilar input variables (at different measurement sites). This approach assumes that the dominant phenomena to be predicted are the relationships between the variables and that temporal trends are not significant or cannot be observed. The feature set for the "most recent" configuration is shown in Table 6.5. Several general-purpose algorithms are applied and the performance results are shown in Table 6.4. While this is a regression problem, some tree-based algorithms such as decision trees also perform well for some regression domains. We provide results from the Reduced Error Pruning Tree (REPTree, [Witten and Frank, 2005])

offset.

algorithm, a variant of decision trees, as an example of this approach, but the performance results are disappointing: this is possibly due to the method not encoding the underlying phenomena well.

		Target Offset					
Method	**Algorithm**	12	24	48	72	96	120
mostrecent	REPTree	73.91	77.42	99.87	138.74	182.40	161.62
mostrecent	PLS	16.94	35.12	61.18	72.94	82.31	92.83
mostrecent	Ridge	16.94	35.12	61.18	72.94	82.31	92.83
mostrecent	nuSVR	12.38	26.20	50.79	66.71	79.42	92.94
mostrecent	skRF	67.91	63.64	92.56	126.16	147.15	152.32

Table 6.4: Test set flow RMSE for multivariate methods. **Bold** values show error statistics that are smaller than the "last value" method benchmark.

	Lagged Offsets (hours)														
Class	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80	-88	-96	-104	-112
1	•														
2	•														
3	•														
5	•														
6	•														
7	•														

Table 6.5: The "most recent" lag scheme for multivariate methods.

6.2.3 Multivariate Methods with Temporal Aspect

To incorporate trends in the multivariate model, time lagged variables can also be added to the feature sets. Using a regression model as a time series model using lagged variables is commonly referred to as a "tapped delay line" (in an engineering context, [Sauer, 1994]) or a "state-space reconstruction" (in a physics context, [Kugiumtzis, 1996]). The most basic multivariate configuration with lags corresponds to the lag scheme configuration in Table 6.7 which has 15 time lagged instances of each of the 6 feature classes for a total of 90 variables in the augmented feature vector. For convenience, we refer to this as a Full Lag Scheme ("fsAll"). The performance results (Table 6.6) of this scheme are not found to improve over the "last value" configuration for any time horizon. This may suggest that these data are not informative for the prediction task or that the calibration process (fitting a model to the training data) is inhibited by the large number of variables in the augmented feature set. If the latter, an explicit feature selection step may ameliorate this difficulty.

		Target Offset					
Method	**Algorithm**	12	24	48	72	96	120
fsALL	REPTree	68.00	73.69	147.77	154.45	126.74	134.60
fsALL	PLS	16.92	26.27	52.60	89.55	124.38	154.21
fsALL	Ridge	22.82	50.59	119.77	213.76	310.77	372.29
fsALL	nuSVR	14.56	31.25	62.73	116.77	183.07	249.26

Table 6.6: Test set flow RMSE for multivariate methods with temporal data. **Bold** values show error statistics that are smaller than the "last value" method benchmark.

	Lagged Offsets (hours)														
Class	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80	-88	-96	-104	-112
1	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
2	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
3	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
5	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
6	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
7	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

Table 6.7: The "fsALL" (Full) Lag Scheme for multivariate methods.

6.2.4 Data Driven Feature Selection with Temporal Aspect

The large number of features used as input in these experiments can make prediction performance poor for many algorithms. A common approach to alleviate this challenge is to use a fully-automated feature selection algorithm to prune the feature space which can improve prediction performance. Two common methods are applied here: one filter-based (Correlation-based Feature Selection) and one wrapper-based (Best-first Search Feature Selection). A feature selection algorithm is expected to generally improve the prediction results in large multivariate models when compared to the same model built without feature selection as a preprocessing step. In practice, for this domain, this improvement is not always observed.

Briefly, Correlation-based Feature Selection (CFS) is a filter-based method that uses Pearson's correlation. Specifically, it is a greedy algorithm that seeks to maximize the mean target-to-feature intercorrelation divided by the mean feature-to-feature intercorrelation. In regression domains such as this, it is likely that small changes in the value of individual features will not generate large effect the value of the target (i.e. no non-linear effects), so it is unlikely that there will be large variations in the target-to-feature intercorrelation values (i.e. many features will look similarly correlated with the target). The underlying relationship is likely to be a weighted linear sum. This is based on an idealized physical model of the domain: if only fixed (non-varying) flows are considered, the sum of flows for all upstream tributaries should be roughly equal to the flow at a downstream site with variances due to evaporation, water use, and physical

variations. Overall, the experimental performance of CFS is not very promising regardless of the underlying ML algorithm.

Best-first Search Feature Selection (BFS) is a wrapper-based method that uses iterative improvement of the feature subset and the underlying learning algorithm to develop a good performing feature subset. BFS performs best among the data-driven feature selection algorithms but it is sensitive to the choice of ML algorithm as shown. As BFS is a greedy search strategy, it is not guaranteed to find the best solution within the search space. Also, the search space for this domain is large and contains 90 lagged variables for $2^{90}(\approx 10^{27})$ possible configurations[5].

Results using multivariate models and data-driven feature selection methods are shown in Table 6.8. CFSF refers to CFS run using a forward search process (i.e., initializing the search with an empty feature set and iterating by adding features until the stopping criteria is met). CFSB and BFSB refer to the two respective algorithms run in backward search mode (i.e., starting from the set of all features). The CFSF experiments compute the feature set without use of the underlying machine learning algorithm, so those experiments all use the same feature set for the same target offset across all choices of machine learning algorithm.

The CFS with PLS results are best in these experiments and have lower error than the "fsALL" experiments for the PLS machine learning algorithm. The results for Ridge Regression are similar to the scores of PLS so these are not included for compactness. The results of nuSVR are also similar to PLS but the computation time is prohibitively large due to the large number of variables (due to the many instances of lagged variables), so these are also not included in the remaining experiments involving many feature selection evaluations.

Another approach to improving performance is to add more informative variables (e.g. precipitation information). Toward that approach, we add new features in the next section.

		Target Offset					
Method	**Algorithm**	12	24	48	72	96	120
BFSB	PLS	47.51	85.70	88.21	134.76	223.24	215.37
BFSF	PLS	201.10	200.83	200.35	199.97	199.66	199.44
CFSB	PLS	29.62	34.63	59.25	71.33	97.67	115.99
CFSF	PLS	40.69	36.19	73.11	83.56	99.73	118.11
BFSF	REPTree	85.99	88.18	93.56	163.21	155.57	157.64
CFSF	REPTree	51.70	83.04	101.37	114.56	111.86	140.59

Table 6.8: Test set RMSE for multivariate methods with data driven feature selection.

[5]This count corresponds to 7 observation sites times 15 possible lags at each site.

6.2.5 Precipitation Data in Prediction

As the underlying phenomena observed in this domain is greatly effected by precipitation in the river network region, adding precipitation predictions to the input feature vector may improve the prediction results. Predicted precipitation is available for each measurement site in the following format from the NOAA observations: there is a total precipitation prediction for each 6 hour window from the current time to 60 hours into the future, for a total of 10 additional variables. The feature class for the precipitation prediction at a specific measurement site contains 10 variables. The raw prediction data for each 6-hour time window is given as discrete values in inches: $\{0.0, 0.1, 0.25, 1.0\}$.

Adding the precipitation variables (and the corresponding lagged variables) increases the feature vector to 282 lagged variables in the Full Lag Scheme (corresponds to Table 6.9). The performance of the multivariate methods both with and without feature selection is shown in Table 6.10. Comparing the experiments both without (Table 6.6) and with (Table 6.10) precipitation data reveals that the addition of the precipitation variables does not generally improve the prediction results for most configurations. Including the precipitation data increases the model calibration time for many of the methods (nuSVR's calibration time increases by 3 times from approximately 1200 seconds to over 3600 seconds). More detailed statistics (validation set error, feature set size, model run time) of these experiments can be found in the additional river flow experimental results section of Appendix B.

	Lagged Offsets (hours)														
Class	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80	-88	-96	-104	-112
1	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
2	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
3	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
5	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
6	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
7	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
11	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
21	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
31	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
51	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
61	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
71	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

Table 6.9: The "fsALL" lag scheme for multivariate methods with precipitation data.

Adding this precipitation data should improve the prediction results from a naïve understanding of the domain. Unfortunately, it instead increases calibration cost and risk of poor predictions due to the addition of many possibly irrelevant or redundant features.

		Target Offset					
Method	**Algorithm**	12	24	48	72	96	120
fsALL	REPTree	68.28	73.62	147.27	154.17	126.74	134.53
fsALL	PLS	16.92	26.27	52.60	89.56	124.38	154.20
fsALL	Ridge	29.88	53.05	91.66	127.69	237.70	399.05
fsALL	nuSVR	39.97	69.54	100.51	119.52	157.05	224.30

Table 6.10: Test set RMSE for multivariate methods with data driven feature selection and precipitation data.

6.2.6 Developer-Guided Feature Selection (DGFS)

The generated models constructed so far utilize only information from the data stream itself without leveraging any feature relationships that may be known to the model developer. By using very basic domain knowledge of the physical feature relationships to augment feature selection using DGFS, it may be possible to improve prediction performance.

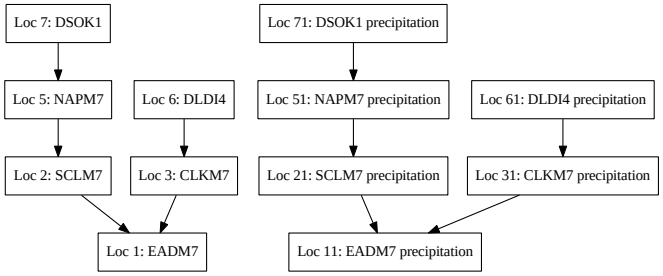


Figure 6.5: Constraint relationships between measurement sites for both flow and precipitation measurements. The node text contains the feature class number and the measurement site name.

For this domain, we provide simple physical constraints derived from the river net work in Figure 6.5. The constraints on the time lags for feature classes are expressed mathematically in Table 6.1. Please note that the feature classes of the precipitation data have no constraint relationships with the water flow feature classes. It is not clear at the outset what constraints would be appropriate, so to be conservative, the relations between precipitation feature classes and flow feature classes are left unconstrained, just as a novice practitioner may decide.

The DGFS methodology described in Chapter 3 is utilized to determine the accuracy scores using several feature selection methods including exhaustive (in non-precipitation experiments only), Greedy, Random, and Guided. A parallel network of feature classes is added to the lag scheme constraints as shown in Figure 6.5. This parallel network increases the search space of lag scheme configurations, but the search is still feasible for the non-exhaustive feature selection approaches (Greedy, Random, and Guided). The exhaustive (non-precipitation) experiment

Method	Algorithm	Target Offset					
		12	24	48	72	96	120
NOAApredictions	Not Applicable	**10.04**	**11.67**	**17.49**	**24.40**	**33.66**	**45.92**
fsExhaustive	PLS	**10.70**	**19.31**	**36.21**	**53.22**	**69.44**	**84.46**
fsGreedy	PLS	**12.01**	**22.27**	**44.15**	**59.89**	**67.82**	**81.58**
fsRandom	PLS	**11.91**	26.33	54.88	**62.54**	**76.60**	**91.61**
fsGuided	PLS	13.40	25.36	**41.88**	**53.22**	**67.82**	**88.57**
precip+fsGreedy	PLS	**10.93**	**20.56**	**37.87**	**56.73**	**67.74**	**76.26**
precip+fsRandom	PLS	**12.19**	**23.11**	48.96	64.67	**77.02**	**89.17**
precip+fsGuided	PLS	**10.55**	**21.56**	**36.70**	**61.00**	83.52	**77.13**

Table 6.11: Test set RMSE for DGFS methods with and without precipitation feature classes. Values shown in **bold** indicate this experiment outperformed the score for the "last value" experiment for the prediction horizon.

required 572387 evaluations while the non-exhaustive versions used far fewer and still achieved competitive results: Greedy, 113 iterations (on average); Guided, 1400 iterations; Random, 2800 iterations.

The results of the DGFS experiments are shown in Table 6.11. These feature selection methods outperform the "last value" baseline for many time offsets. As expected, the NOAA hybrid model is consistently best and can serve as a performance target. The with-precipitation experiments generally have improved accuracy. This improvement is most evident in the 120 hour offset experiments. The best methods overall are the Guided and Greedy versions.

To examine the effects of using the precipitation data, we provide a 2-d histogram comparing selected methods for the 48-hour offset prediction target in Figure 6.6.

The future precipitation information reduces the likelihood of large flow underestimates for the with-precipitation version as shown in the prediction error histogram at the top of the figure. The error distribution of Guided with precipitation is centered closer to zero than Guided without precipitation data. The effect of precipitation on the system is lagged (it takes many hours for a precipitation event to increase river flows), so it may not be as informative for near term predictions (i.e. < 72 hours) than for farther-term predictions.

The exhaustive version of feature selection should generally have superior performance compared to the non-complete search types (Greedy, Random, and Guided), and this is observed in most offsets in Table 6.11. The configuration space is the same for all search types, but each method is likely to choose a different optimal model due to differences in each of the search methods. The exhaustive results may not be strictly better than the non-complete search as he best feature set is chosen using the validation set results (which may be different for each method) and the scores shown are for the test set. The validation set scores are shown in the

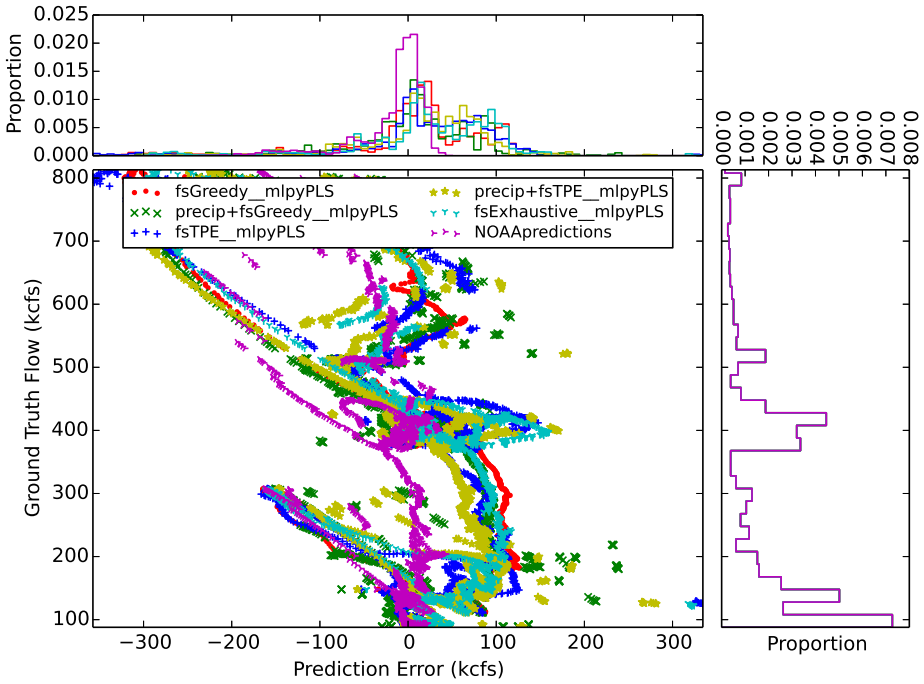


Figure 6.6: EADM7 2-dimensional histogram comparing DGFS methods at 120 hour prediction target.

extended experimental results found in Appendix B.

6.2.7 Statistical Performance Comparison

To understand the differences between the models, we compare the performance for the 48 hour prediction horizon. The best methods found in experiments on this domain are available for comparison in Table 6.12 along with the NOAA model performance as a baseline.

To verify that the prediction models are different from a statistical perspective, we use statistical significance testing to compare the best configurations of each type. Significance results are in Table 6.13 which use the Nemenyi test[6] to determine statistical significance among all pairs.

In investigating the results further, we endeavor to show that these methods represent an improvement and, most importantly, to quantify the performance differences. There has been

[6] A paired statistical test to compare the prediction performance of many methods. If a single paired test such as the Wilcoxon Signed-Rank test is used in a multiple method comparisons setting, it is likely that spurious significant pairs will be found due to randomness and the large number of method pairs. The method has a correction to account for the multiple comparisons problem [Bagheri et al., 2012; Demšar, 2006; Zimmerman and Zumbo, 1993].

Rank	Name	Score	(Std. Dev.)	Count
1	NOAApredictions	17.48982	(13.19276)	2931
2	fsExhaustive_PLS	36.20544	(24.78817)	2931
3	precip+fsGuided_PLS	36.70078	(25.34288)	2931
4	precip+fsGreedy_PLS	37.86815	(26.66006)	2931
5	fsGuided_PLS	41.88048	(31.89135)	2931
6	fsGreedy_PLS	44.14797	(30.81353)	2931
7	precip+fsRandom_PLS	48.96109	(32.93030)	2931
8	fsRandom_PLS	54.87830	(36.63715)	2931

Table 6.12: River flow performance statistics (prediction error mean, standard deviation, and count) for 48 hour ahead predictions.

Rank	Name	Rank							
		0	1	2	3	4	5	6	7
0	NOAApredictions	–	◊-31.0	◊-30.5	◊-28.8	◊-25.5	◊-37.2	◊-48.1	◊-51.6
1	fsExhaustive	◊31.0	–	0.4	2.1	◊5.4	◊-6.2	◊-17.1	◊-20.6
2	precip+fsGuided	◊30.5	-0.4	–	1.7	◊4.9	◊-6.7	◊-17.5	◊-21.1
3	precip+fsGreedy	◊28.8	-2.1	-1.7	–	◊3.2	◊-8.4	◊-19.3	◊-22.8
4	fsGuided	◊25.5	◊-5.4	◊-4.9	◊-3.2	–	◊-11.7	◊-22.5	◊-26.1
5	fsGreedy	◊37.2	◊6.2	◊6.7	◊8.4	◊11.7	–	◊-10.8	◊-14.4
6	precip+fsRandom	◊48.1	◊17.1	◊17.5	◊19.3	◊22.5	◊10.8	–	◊-3.5
7	fsRandom	◊51.6	◊20.6	◊21.1	◊22.8	◊26.1	◊14.4	◊3.5	–

Table 6.13: Statistical significance testing for all method pairs at 48 hour ahead predictions using the Nemenyi Test. The value of each cell is the critical distance of the row/column pair. A "◊" in a cell indicates that the pair has a statistically significant difference (i.e. the row method is statistically different from column method at the 95% significance level). The absence of a "◊" for a pair indicates that the null hypothesis cannot be rejected.

much discussion of the direct application of statistical significance testing on data with large (> 1000) sample sizes [Gigerenzer, 2004; Lin et al., 2013]. To address this concern we also present the alternative approach of measuring the confidence interval of the performance difference on a pairwise basis Johnson [1999]. We estimate the 95% confidence interval for the difference in prediction error between methods in Table 6.14 for prediction offset of 48 hours. Each cell shows the confidence interval estimate of the mean difference in performance.[7] This measurement is

[7] A -0.5 to -0.25 value indicates the row method produces a mean error 0.5 to 0.25 units lower than the column method. Negative values indicate an improvement; positive values indicate a degradation.

Rank	Name	Rank							
		0	1	2	3	4	5	6	7
0	NOAApredictions	–	(−13.01, −12.01)	(−13.01, −11.51)	(−13.51, −11.51)	(−12.51, −11.01)	(−18.51, −16.51)	(−21.51, −20.51)	(−28.51, −26.01)
1	fsExhaustive	–	–	(−0.61, 0.04)	(−1.11, 0.39)	(0.44, 1.34)	(−6.21, −4.11)	(−7.46, −5.96)	(−14.01, −11.51)
2	precip+fsGuided	–	–	–	(−1.41, −0.16)	(0.94, 2.19)	(−6.16, −4.21)	(−8.36, −6.86)	(−14.51, −12.01)
3	precip+fsGreedy	–	–	–	–	(0.39, 1.74)	(−5.71, −4.46)	(−8.31, −6.71)	(−14.01, −12.01)
4	fsGuided	–	–	–	–	–	(−5.31, −3.56)	(−9.21, −7.91)	(−14.01, −11.51)
5	fsGreedy	–	–	–	–	–	–	(−4.46, −2.06)	(−9.01, −7.96)
6	precip+fsRandom	–	–	–	–	–	–	–	(−5.81, −3.46)
7	fsRandom	–	–	–	–	–	–	–	–

Table 6.14: Accuracy improvement effect size confidence intervals (95% significance) for pairwise method comparisons. Each value tuple indicates the 95% confidence interval for the mean error of the row method versus the column method. Values are in kcfs. Negative values indicate the row method has lower error than the column method.

commonly called the "effect size" in the statistics literature.

6.2.8 Mistakes in DGFS: Bad Constraint Relationships

The performance of DGFS for prediction depends to some extent on the quality of the feature classes and constraints provided by the model developer. This section seeks, through experimentation, to show the performance degradation caused by constraints that do not match the underlying physical relationships in the domain.

The known correct feature class constraints (A) for this domain along with two "bad" constraint graphs (B and C) are visualized in Figure 6.7. Network A is a visualization the user-provided constraints utilized for all DGFS experiments so far in the chapter. Network B generates a total order from the partial ordering of graph A. Network C has many incorrect constraints when compared to the physical configuration of the measurement sites and represents a worst case scenario for user-provided constraints. The DGFS framework is run using these incorrect constraint graphs to determine their effect on accuracy.

Using these badly constructed constraint sets, another set of experiments is run. The results are provided in Table 6.15. The correct constraint network still provides the best performance in almost all cases. The bad constraint networks (B and C) reduce very poor results for some experiments. The physically correct constraint network seems to provide better results with less chance for large errors. These mistakes are examined further in the next section.

In examining the results further, we can inspect the chosen lag schemes for each constraint

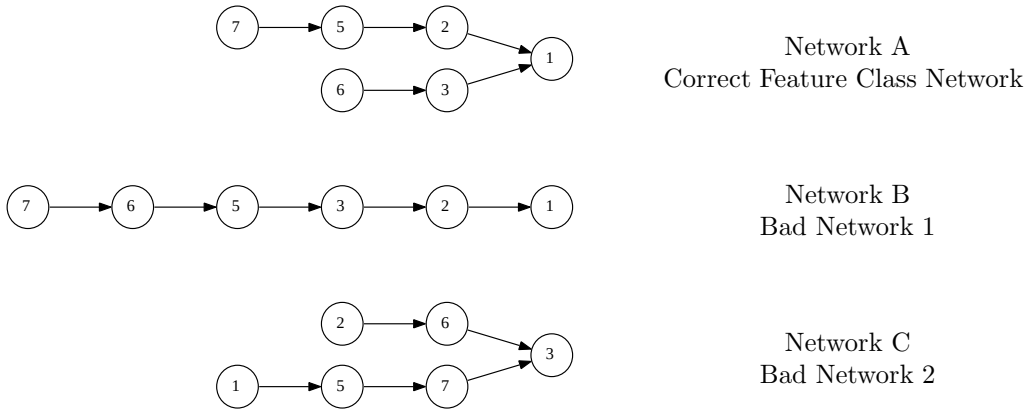


Figure 6.7: Feature class constraint networks for river flow domain.

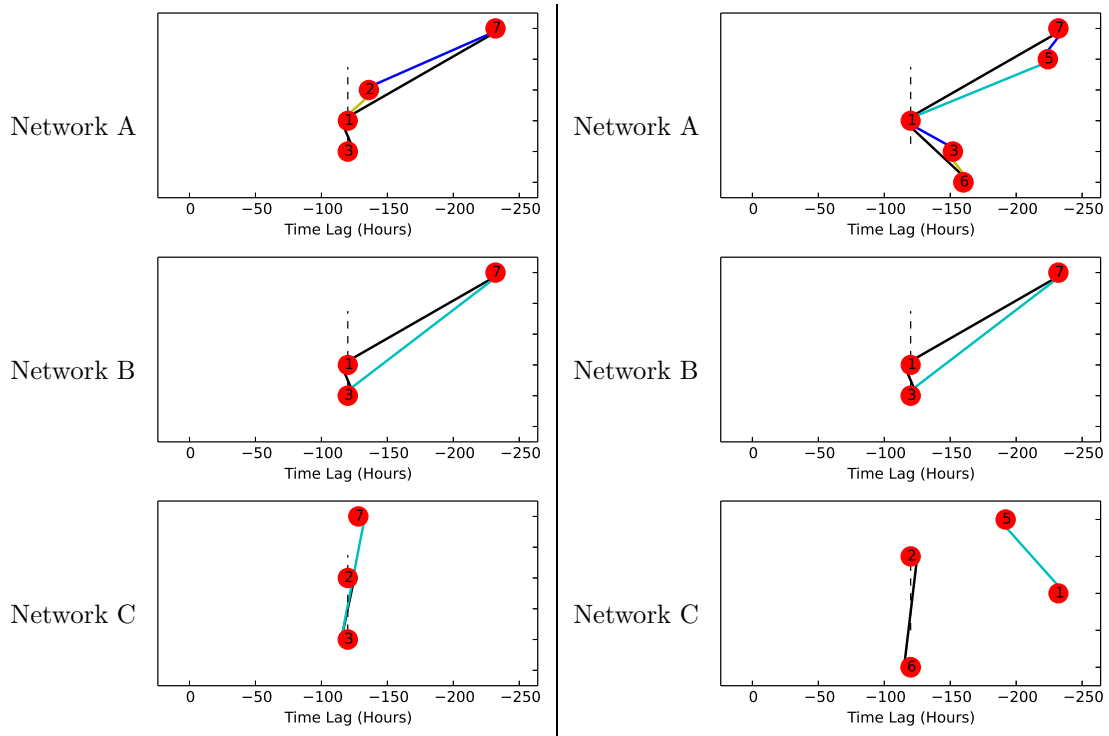
network in Figure 6.8. The Network A results for both exhaustive and guided search methods is most similar to the ground truth relationships (visualized in Figure 6.9). Network B somewhat similar to the ground truth, but even fewer features are used. Network C models are most dissimilar from the ground truth for all search strategies. In comparing the search methods for the same constraint network, there are also patterns. The exhaustive search tends to produce more "minimal" models than the other approaches in that fewer features are used in the models. The guided approach uses many features but the assigned time lags for each feature class are congruent with the relationships in the ground truth scheme.

It is important to note that the chosen time lags are often significantly offset from the ground truth relationships but the predictions may be useful because of the periodic nature of the data. For a complete listing of the best lag scheme configurations found in the experiments covered in this chapter, consult the Appendix B.

In the well documented domain of river flow, it is unlikely that mistakes in constraint encoding would be made by the model developer, but in applications where the relationships are less clear, this is possible. For example, in another domain such as a sensor network containing many sensors with changing relationships, constraint encoding mistakes are more likely.

		Target Offset					
Network	**Method**	12	24	48	72	96	120
–	NOAApredictions	**10.04**	**11.67**	**17.49**	**24.40**	**33.66**	**45.92**
A	fsExhaustive	**10.70**	**19.31**	**36.21**	**53.55**	**69.44**	**84.46**
A	fsGreedy	**12.01**	**22.27**	**44.15**	**59.89**	**67.82**	**81.58**
A	fsGuided	13.40	25.36	**41.88**	**53.22**	**67.82**	**88.57**
A	fsRandom	**11.91**	26.33	54.88	**62.54**	**76.60**	**91.61**
B	fsExhaustive	66.43	65.31	67.52	67.81	79.59	**88.20**
B	fsGreedy	**11.93**	**22.29**	**42.51**	**61.11**	**78.95**	**85.21**
B	fsGuided	13.31	30.03	**41.43**	84.91	100.15	**84.90**
B	fsRandom	14.44	29.38	56.77	100.10	118.89	99.47
C	fsExhaustive	66.43	65.31	67.52	67.81	79.59	**88.20**
C	fsGreedy	48.28	50.86	55.45	**58.57**	**68.93**	**79.46**
C	fsGuided	**11.93**	24.86	**42.51**	**58.92**	**68.35**	**84.47**
C	fsRandom	27.24	39.20	61.34	74.05	89.47	98.40
–	lastvalue	**12.92**	**24.06**	**44.81**	**63.19**	**79.09**	**92.70**

Table 6.15: Validation set RMSE for DGFS experiments using the PLS learner with different constraint networks (A, B, and C) and comparing with benchmarks ("last value" and "NOAA Predictions"). **Bold** values show error statistics that are smaller than the "last value" method benchmark.



(a) Exhaustive Feature Selection Search

(b) DGFS Guided Feature Selection Search

Figure 6.8: Comparing best lag schemes for 120 hours ahead predictions found on constraint networks A, B, and C.

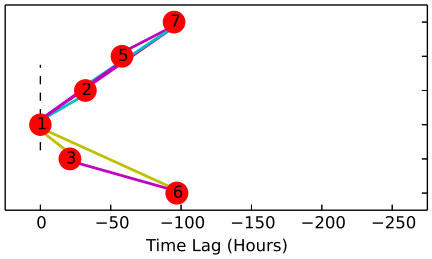


Figure 6.9: A lag scheme visualization generated using the ground truth correlation measurements. Edges in the graph denote constraints derived from the physical relationships.

6.3 Characterizing Constraint Network Quality

To measure the quality of the constraint network we use the concept of *edit distance*. The edit distance is a measure of correspondence between two lag scheme configurations. Mathematically it is:

$$ED(\alpha, \text{offset}) = \left(\sum_{i=1}^{n} \begin{cases} |\delta_i^{GT} - (\delta_i^{\alpha} + \text{offset})| & \text{where } \delta_i^{\alpha} \neq \text{NULL} \\ 0 & \text{otherwise} \end{cases} \right) \div \left(\sum_{i=1}^{n} \begin{cases} 1 & \text{where } \delta_i^{\alpha} \neq \text{NULL} \\ 0 & \text{otherwise} \end{cases} \right) \tag{6.2}$$

In Equation 6.2, GT is the ground truth lag scheme, α is the candidate lag scheme, offset is the prediction horizon in hours, and $\delta_i^{(x)}$ is the lag offset for feature class i in lag scheme configuration (x). $ED()$ measures the mean time offset distance for all feature classes assigned in the lag scheme. This measures the correspondence between individual lag scheme configurations discovered through feature selection search and the ground truth physical relationships (i.e., lower is better, closer to the ground truth).

Given the observed correlations between gauge sites, the physically correct lag scheme can be computed for this domain. This configuration is shown in Table 6.16. Because the measurements are hourly and, for efficiency, the available time lag assignments are quantized on 8 hour offsets, the closest scheme does not correspond exactly to the ground truth configuration. The edit distance between the ground truth correlation measurements and this scheme is 1.17. For ease of understanding, a visualization of the Table 6.16 configuration is available in Figure 6.9.

	Lagged Offsets														
Class	0	-8	-16	-24	-32	-40	-48	-56	-64	-72	-80	-88	-96	-104	-112
1	•														
2					•										
3				•											
5								•							
6													•		
7													•		

Table 6.16: The closest lag scheme table subject to the 8 hour time offsets used in the experiments. The edit distance between this and the ground truth network is 1.17.

The edit distance measure can be used as a proxy for the "correctness" of a lag scheme configuration. We define lag scheme complexity as the number of variables or assigned feature classes. We hypothesize that, for the same level of lag scheme complexity, models having a lower edit distance will have better accuracy. This idea can be used for examining the performance differences for different feature class constraints. The 3 competing sets of feature class constraints (Correct, Bad Network 1, Bad Network 2) provided in Figure 6.7 differ significantly in the edit distance values generated. In the search phase of DGFS, many candidate lag schemes are

evaluated for each model. For each of the 3 constraint sets, the distribution of validation set errors can be plotted over the range of edit distance values observed. Figure 6.10 shows the validation set error distributions for the Random and Guided search modes on configurations with lag scheme complexity equal to 6.

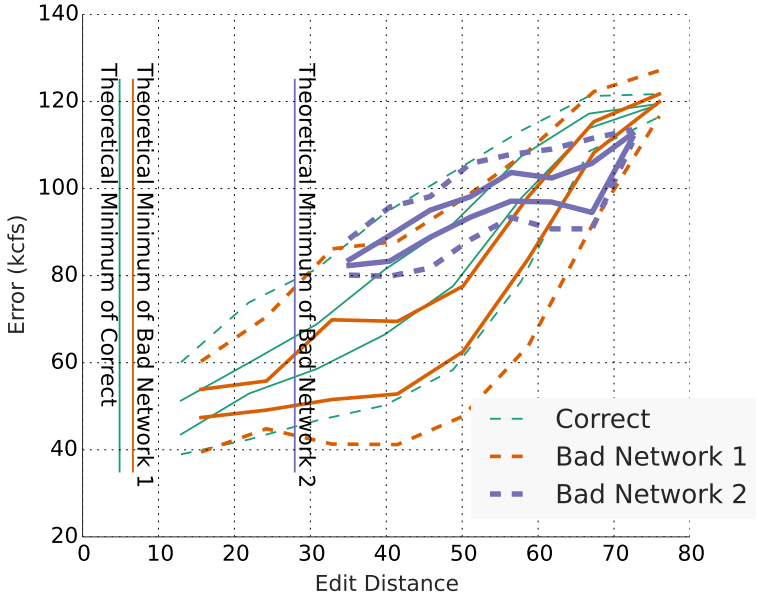


Figure 6.10: Edit distance vs. mean prediction error distributions for 24 hours ahead test set predictions. The quintiles plotted (as percentiles 20, 40, 60, and 80) show the distribution of scores for the edit distance value on the x-axis.

The minimum possible distance values vary based on the constraint network, prediction horizon, and the ground truth physical relationships. For 24 hours ahead predictions the minimum edit distance for each network against the ground truth correlations is: 5.1 for Network A, 6.5 for Network B, and 27.5 for Network C. The lower bound values for each constraint set are plotted in Figure 6.10. The figure shows there is a trend between a candidate's edit distance and its accuracy for all 3 constraint sets. Another way to say this is that: when sampling from the search space of possible feature sets, it is better to sample configurations with smaller edit distances. Also, when choosing a constraint set, constraint sets that permit feature sets with low edit distance (i.e. Network A) tend to have better accuracy than constraint sets with larger edit distances (i.e. Networks B and C). It is still possible for a network with larger edit distances to find a low-error model, but the distribution of scores for larger edit distance configurations is wider. Succinctly, a lower edit distance of a feature set seems to be a necessary requirement

for low error but it is not sufficient to guarantee low error.

6.4 Conclusions

DGFS is a technique for biasing the feature selection process, and is therefore a technique for biasing model calibration for machine learning. The bias imparted by this proposed method is easily understood by the model developer. Some advantages are: 1) the developer can quickly "sanity check" the resulting model by examining the chosen lag scheme, and 2) the developer can adjust the level of bias as needed for other reasons (such as to reduce model calibration time). The constraint network provided and the feature selection search method choice together determine the run time. If run time is not constrained, the exhaustive feature selection search is preferred for achieving lowest error for this domain. In run time constrained applications, faster search process (Greedy or Guided) can efficiently find feature set configurations without a large degradation from the exhaustive approach.

Chapter 7

Applied Domain: Supply Chain Price Prediction

In this chapter, the proposed feature selection approach is used in the construction of price prediction models in a complex supply chain domain from a manufacturer's perspective. For this domain, we first present a data extraction method that is able to leverage information contained in the movements of all variables in recent observations. The extracted data is then used to build price prediction models that incorporate all observable variables in the domain from the manufacturer's perspective. To control model complexity and improve prediction performance, our proposed Developer-Guided Feature Selection method is used to bias the prediction model generation. This work is experimentally validated with data from a competitive multi-agent supply chain setting, the Trading Agent Competition for Supply Chain Management (TAC SCM). Our method achieves competitive (and often superior) performance compared to the state-of-the-art domain-specific prediction techniques used in the 2008 TAC SCM Prediction Challenge competition.

7.1 Price Prediction in TAC SCM

TAC SCM is a complex, heterogeneous-agent supply chain game that is designed to simulate an oligopoly market of competing computer manufacturing agents who must autonomously purchase component parts, manufacture computers, and sell the finished computers to end customers over a simulated product lifetime of one year (220 simulation "days"). Each run of the simulation, which takes approximately one hour in real-time, is unique because the market's behavior will vary both due to changes in the underlying market conditions (available component

supply and customer demand), and due to changes in the behavior of individual agents.

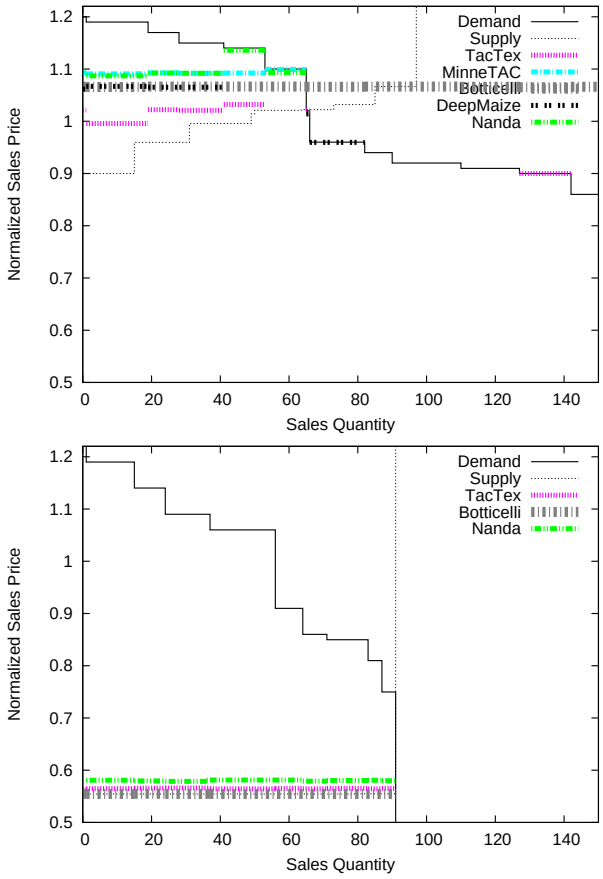


Figure 7.1: Customer market supply and demand for a product on day 4 (top) and day 89 (bottom) of game tac01-#3454. The demand line is the total demand. The dotted lines correspond to the offers made by each agent and the height of the line indicates the offer price. The agent with the lowest offer price for a given request will receive the order.

Some of the research we have done has resulted in a large body of visualization tools and key performance indicators relevant for TAC SCM [Groves et al., 2010, 2014]. An example is presented here to show insights into the domain found through visualization. Figure 7.1 visualizes supply and demand pressures for two individual days in the component purchase market. The supply line (thin dotted line) represents the ascending price-sorted set of offers made by the agents in response to customer requests. The total demand from customers is shown in the demand line (solid black line), sorted by decreasing reserve price. Offers from individual agents are visible as short horizontal segments set vertically at the price specified in the offer. The offer with the lowest price becomes the accepted order for each request. Using this graph

it is possible to estimate the current market pressure based on how the bids are compressed in terms of price (y-axis). The lower plot has much higher market pressure because the bids from all agents are compressed in a very narrow price range.

7.1.1 The Prediction Problem

The basic purpose of prediction in TAC SCM is to provide information about the future that is relevant to decisions (i.e. supply purchasing, manufacturing, sales bidding) that must be made now. In this supply chain scenario, prediction is concerned with providing information about future costs, prices, and demand so that decisions can be made today that maximize future profit. Other aspects such as limited information, informational delays, changing conditions, and observation noise can add complexity and make prediction even more difficult. TAC SCM is a market simulation game that possesses all these challenges.

To limit the scope of this chapter, we are primarily concerned with the price prediction aspects of the competition. A subcompetition called the "Prediction Challenge" was initiated in 2007 to isolate this aspect of the simulation for study [Pardoe and Stone, 2008]. This subcompetition facilitated direct comparisons of prediction performance on four prediction types: (1) current product prices, (2) future product prices, (3) current component prices, and (4) future component prices. "Current" refers to predictions about prices revealed to an agent on the next day; "future" refers to predictions about prices revealed to an agent an additional twenty days in the future from the current day.

The challenge of predicting in this environment is that many of the variables have relationships to other variables that are not easily quantified, can vary due to the agent population, and can change over time. Economic theory suggests that there is a direct relationship between the cost of parts and the cost of finished goods, but, in a situation where parts are shared across many products, the relationships are dynamic. Instead of modeling these relationships individually, we use regression to model the relationship between observable inputs (prices and quantities) and future prices. This model is able to implicitly determine correlations between input variables based on historical data. To effectively model these relationships, a corpus of representative historical data is required to calibrate the model. The observations used to construct the training set features should be representative of the range of scenarios the model will need to predict over. In a multi-agent setting, this means that the training data should be generated by a similar set of opponents as when predictions are to be made.

TAC SCM manufacturing agents operate as buyers in the component supply market and as sellers in the finished goods market. To make the market clearing consistent, each successful transaction negotiation consists of a sequence of 3 messages: RFQ (request for quote), offer, and order. The data fields for each message type is shown in Figure 7.2. The major difference

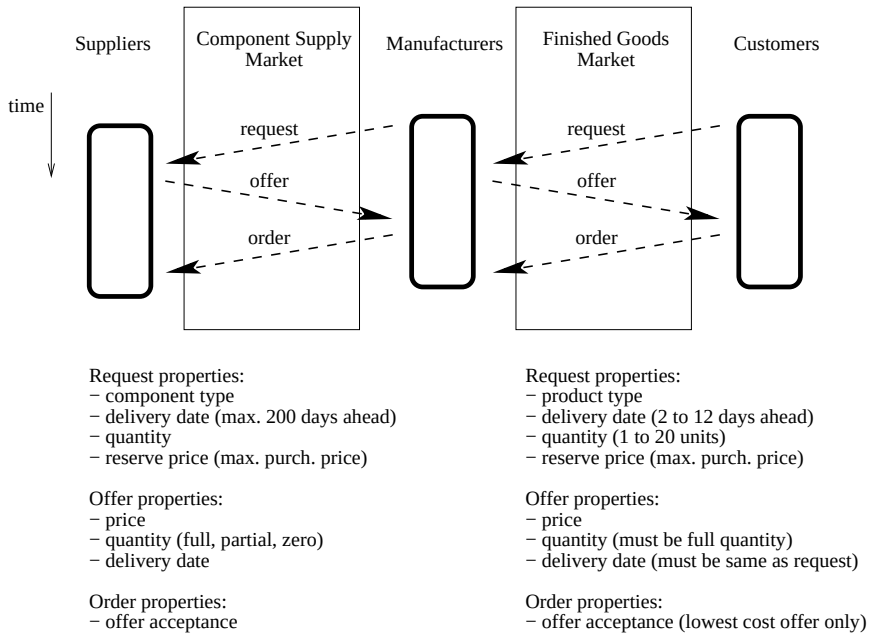


Figure 7.2: Transaction negotiation mechanisms in both the component and finished goods markets for TAC SCM.

between the component supply market and the finished goods market is in the range of lead times for the requests: the component supply market has a maximum manufacturer specified lead time of up to 200 days. While in the finished goods market, products are requested by customers at most 12 days in advance. This mismatch in the planning horizon of the two markets is a primary motivation for prediction in the scenario.

Dynamic prediction is crucial for any agent operating autonomously. The majority of prediction models can be classified as one of two types: on-line (only uses information from a short window of recent observations) and off-line (significant volumes of historical data are used to calibrate the model). While on-line methods can be more robust to changes in the environment and can provide good performance in a dynamic environment, often better performance can be achieved by incorporating training on historical data. For these reasons, typically on-line models are tried first, and off-line models are used as higher performance becomes necessary.

The TAC SCM competition as well as other autonomous agent competitions involving travel brokering [Stone and Greenwald, 2005] and stock trading (i.e. Penn Lehman Trading Competition [Kearns and Ortiz, 2003]) require agents to do significant forecasting to achieve consistent, strong performance. The methods that employ information learned from historical data are the most effective.

Current and future prices of products. The prediction methods employed by top performers in the TAC SCM competition often involve a combination of offline training on historical data for computation of latent (or not always visible) information. In the context of the TAC SCM Prediction Challenge, it is required to predict for the next day and 20 day ahead sale price of each product type. The TacTex agent makes next day product price predictions using a particle filter to compute the lowest price offered by any agent in response to each request [Pardoe and Stone, 2007]. The Deep Maize agent uses a k-nearest neighbor technique over a large set of historical data. The feature vector, over which similarity is computed, includes current day estimates of supplier capacity (which features prominently in the pricing equation), today's observed customer demand, and recently observed computer prices [Kiekintveld et al., 2009]. This method explicitly limits the contribution of individual games by using only the most similar observation from each game in the dataset. The CMieux agent uses a decision tree variant called a distribution tree that tracks several distinct price distributions and chooses the most relevant Gaussian normal distribution to use based on publicly known attributes of the price to estimate [Benisch et al., 2009]. A classification-based approach is used by [Ketter et al., 2009] which involves the clustering of observed market behaviors into economic regimes to build both improved current and future product price distribution models. Overall, there are significant tactical reasons to estimate next day prices due to the need to assign a specific bid price to each customer request.

For 20 day ahead sales price predictions ("future product"), most agents report to use methods similar to their next day approaches. The TacTex agent uses the result of its next day particle filter and augments this value by computing an adjustment for 20 day ahead predictions using an additive regression model. The additive regression model estimates the difference between the next day price and the 20 day ahead price and is derived from a set of over 31 visible features (16 product prices, 3 demand signals, and 12 unique component prices) from the current day's observations. This technique is most similar to the PLS regression method discussed in Section 7.1.2 but differs by not using information observed prior to the current day. Deep Maize estimates future product prices using the same k-nearest neighbor used in next day predictions but instead trained on 20 day ahead observations.

Some agents may not be estimating future product prices (20 day ahead predictions) at all because it is not a tactically necessary prediction (agents are not required to compute this value when operating in this environment). Instead, this prediction class may have mainly strategic "long-term" uses for planning long lead-time procurement.

Current and future prices in the component market. Component pricing is significantly more complex due to the need to consider long lead times. When an agent makes a request for component parts it must also specify a delivery lead-time. This is critical because the prices

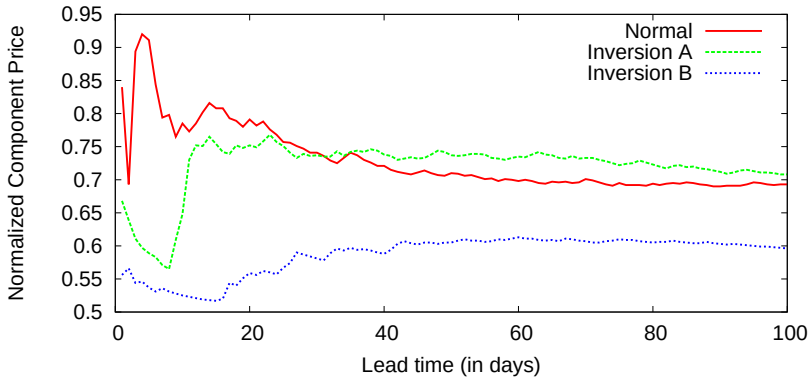


Figure 7.3: Three snapshots of the price vs. lead-time relationship for a single component. These relationships will change daily both due to stochastic elements and agent behavior.

quoted vary significantly over the range of possible lead times. Figure 7.3 shows three example price vs. lead-time plots taken in an individual TAC SCM game and motivates the need for agents to track component prices also in terms of lead-time. The general trend is for prices to be highest in the short term (series "Normal"), but there are instances (series "Inversion" A & B) where short term prices are, in fact, lower. Interestingly, the lead-time aspect of product prices is not explicitly considered by many TAC SCM agents.

TacTex predicts next day component prices with a domain-specific approach by estimating the available production capacity of each supplier. This is done using knowledge of how component prices are calculated by the game server. The fraction of unallocated supplier capacity between the request date and the delivery date is directly proportional to the quoted per unit cost for each request. This value can be estimated by observing differences in the recent quoted prices. This approach cannot be generalized to other domains. For future component requests (20 days ahead), TacTex agent again employs additive regression to learn an adjustment between the current day's price and the future price (20 day ahead). Their algorithm is otherwise unchanged from the next day method.

Deep Maize uses a linear interpolation over recent samples to estimate the component price for a specific due date. An adjustment to the linear interpolation is computed from a Reduced Error Pruning Tree (REPTree), a decision tree variant. The REPTree offset values are trained offline using historical data of next day predictions. This same method is used for 20 day ahead predictions [Kiekintveld et al., 2009].

The CMieux agent uses a k-nearest neighbor approach to estimate next day component costs. The contribution of the k-nearest neighbors is averaged using inverse distance weighting, and distance refers to the temporal difference between the due date requests [Benisch et al., 2009].

7.1.2 Our Approach

The prediction model that PLS regression computes for a variable y is a weighted linear function in terms of the feature values $x_i, i = 1..m$ (where m is the number of features). Mathematically, the prediction model for a variable y can be expressed as[1]: $y = \hat{b}_0 + \hat{b}_1 x_1 + \hat{b}_2 x_2 + \ldots + \hat{b}_m x_m$ where $\hat{b_i}$, $i \in 0..m$ are the regression coefficients computed in the model calibration stage. This form of linear model is suitable for economic modeling in this context and is discussed next.

A simple model. In an economic supply chain context, this simple linear model maps naturally into the domain: the long-run price of a product should be equivalent to a linear sum of the costs of its constituent parts plus the profit margin taken by the manufacturer. The cost of a component could similarly be computed from this ideal model: the long-run cost of some component should be equivalent to a fraction of the prices of products that use the component.

The ideal model presented here, while intuitively attractive, is not sufficient for modeling prices in TAC SCM for several reasons. First, the model does not directly address lead time effects[2]. Also, particularly in the component market, prices vary significantly by lead time: longer lead time requests typically have lower cost. This mismatch between possible lead times in the two markets drives the need for agents to develop mechanisms for coordination, dynamic planning, and prediction. It is for this reason that the range of possible features to include in our model incorporates price observations for multiple component lead times.

Second, the model does not address trends that can be anticipated by observing changes in price over time. Observation of the prices of an individual product over several past time steps (p_t,p_{t-1},p_{t-2},...) provide information to form a prediction about the next value (p_{t+1}). This is equivalent to a univariate time series prediction. Information from previous time steps is included in our model.

Third, visible non-price information about the environment will also have an effect on prices. For instance, the current bank interest rate (the cost of borrowing), and the current storage cost (the cost of holding inventory) will both have effects on price. This is also included in the model.

Fourth, information about the current market situation also has an effect on price. Information about overall aggregate demand is available to the agents by observing the number of

[1] This is a parsimonious expression of the model. In practice, it may be useful to compute y values in a two step process employing, first, the dimensionality reduction and, second, the regression.

[2] In both the component and product markets, agents must commit to buy components or sell products far in advance of the delivery date. In the component market, delivery lead times can be between 1 and 220 days into the future. When an agent makes a component request, it must also specify the delivery date for the request. In the product market, delivery lead times are between 3 and 12 days into the future. The delivery date is specified in the request from the customer, and the agent with the winning bid must honor it or a significant financial per-unit late fee is imposed.

products requested by customers each day. This information can also be included in the feature vector. In keeping with the domain-agnostic nature of our approach, we assume a linear relationship for the non-price features as well.

Including all observable information that could affect price into the feature vector is an obvious approach. But in practice, this is impractical. First, as the number of features rises significantly, the effectiveness of a prediction algorithm is likely to degrade. Second, there is almost no limit to the number of possible features that can be added. The inclusion of irrelevant (or low informational value) features is a problem that we address next.

	Feature Type	Count	Feature
1	Game Instance	2	– storage cost – interest rate
2	Daily Demand	3	– low market – medium market – high market
3	Price for products	16	– SKU 1 ... – SKU 16
4	Price for parts at LT2[a]	16	– comp100sup1lt2[b] – comp101sup1lt2 – comp110sup2lt2 – comp111sup2lt2 ...
	Price for parts at LT6	16	– comp100sup1lt6 – comp101sup1lt6 ...
	Price for parts at LT10	16	...
	Price for parts at LT20	16	...
	Price for parts at LT30	16	...
	All Features	101	

[a]LT2 denotes a delivery lead time of 2 days from the order date.
[b]Price of component 100 from supplier 1 with a delivery lead-time of 2 days.

Table 7.1: List of all computed features available to an agent.

Input Feature Computation. We now illustrate how several distinct types of data available are aggregated into a feature vector of consistent size. Data observable from an individual agent's perspective consists of four feature types:

1. *Game instance features* include values that remain invariant throughout the simulation instance, such as the bank interest rate and storage cost.
2. *Daily market segment demand* is computed from the total number of product RFQs in each of the three market segment on each day.
3. *Daily price observations for products* are the mean sale prices for each of the 16 product types, as observed from the agent. Days with missing product price observations (no

successful sales for a specific product on a given day) have their values computed using a radial basis function interpolation. No consideration is made for variations in lead-time among product price observations.

4. *Daily price observations for components* are the prices of each of the 16 components, as observed by the agent. The prices are added to the feature vector for a pre-defined schedule of lead times of 2, 6, 10, 20, and 30 days. Days with missing component observations on these lead times will have their values computed using a radial basis function interpolation over observations for the component from the previous 5 days.

These input features combine to produce a vector of 101 features (see Table 7.1). Labeled training instances are generated by computing the matrix of observations for the features and appending the known true value (label) for the regression target. This labeled training set can be read by the PLS algorithm to produce a regression model.

Class	Lagged Offsets					
	0	1	2	4	8	16
P3	•					
P3b	•					
P3c	•					
P2	•					
P4	•					
P4h	•					

Table 7.2: A simple configuration for product prediction. A "•" denotes the feature class for the specified time lag is included.

Lagged Features and Hierarchical Segmentation. Using only the most recent values of the 101 possible features as the entire feature set may provide reasonable prediction results in some domains, but, it cannot predict trends or temporal relationships present in the data. The need to represent temporally-offset relationships motivates the idea of adding time-delayed observations to the feature set as well; we refer to this as the addition of *lagged features*. For instance, if it is known that the cost of a component on day $t-8$ is most representative of the price of a product sold on day $t+1$, the 8 day delayed observation of that component should have a high weight in the model. The time delay could correspond to the delay between persistent changes in the observations and the resulting effect in the mean agent behavior.

For this domain, the Developer-Guided Feature Selection constraints use the assumption that more recent observations are likely to have high informational value for price prediction, but time-delayed features may hold informational value as well (i.e. the environment is not completely stochastic). The lead time between a change in the market and its effect on other prices may be longer than one day, but the minimum possible lead time for this effect is 1 day but could vary due to the design of opponent agents. To cover the possibility of changes in agent

design outside of our control, we keep most recent (current day) observations in the model. For reasons of tractability, features with long time delays are only added after features with shorter delays are added. Even with this constraint, searching for the optimal subset from 101 available features is still an intractably large search space.

To reduce the number of possible configurations of features, we introduce the notion of a hierarchical segmentation of the feature set. In this domain, each of the 101 features is placed into one of several classes based on its relationship to the target variable. In cases where a feature belongs to multiple classes, the feature is placed in the class that is most specific. From a minimal amount of domain knowledge derived from the specification document of TAC SCM, we have compiled a class hierarchy for each type of target variable. These hierarchies can be seen for the product prediction and component prediction tasks in Figures 7.4 and 7.5, respectively.

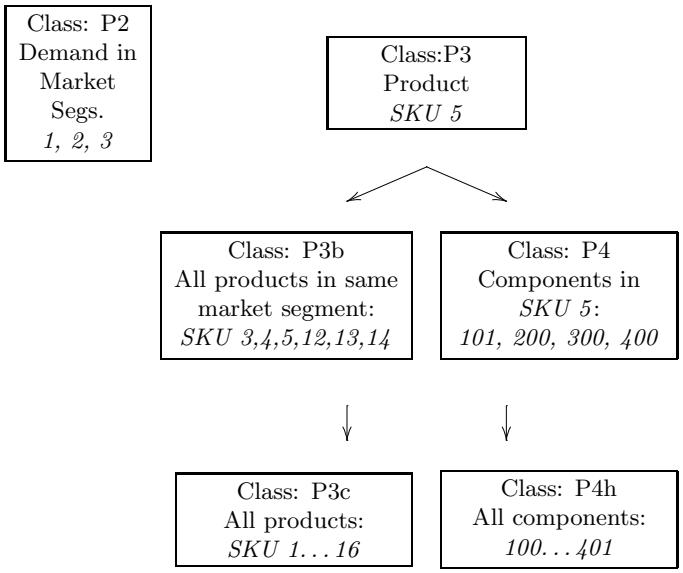


Figure 7.4: Lag scheme class hierarchy for product price prediction. Arrow denotes a *not greater than* relationship (i.e. class P4 should have an equal or lower maximum time offset than class P3).

Thus far we have explained how a better feature set can be chosen based on some knowledge of the domain. Now we will address how to choose the time-delayed data from each class. The simplest lag configuration, shown in Table 7.2, contains the most recent day's value from all feature classes. We posit that time-delayed observations from the variable of interest (*Class P3*) are likely to be predictive as well. Time-delayed observations from other feature classes *may also be* but are less likely to be predictive. It is by this principle that the hierarchy and strict ordering of lagged data additions is based.

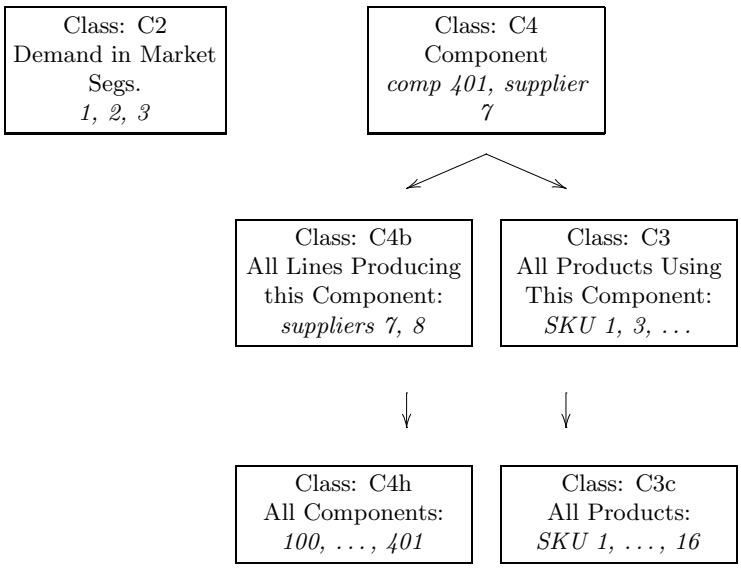


Figure 7.5: Lag scheme class hierarchy for component prediction.

Also, it is necessary to chose the set of permitted lagged offsets from each class. In this domain, two sets of time-delays were found to be useful: time-delays with integer offsets $\{\varnothing^3, 0, 1, 2, 3, 4, 5, \ldots\}$ and time-delays with geometrically increasing offsets $\{\varnothing, 0, 1, 2, 4, 8, 16, \ldots\}$.

We continue with a discussion of the relationships formulated in the class hierarchy for a specific product in this domain, SKU 5. In Figure 7.4, the earlier observations of the variable to be predicted (*Class P3*) are most likely to contain predictive information. Information about other similar products (*Class P4*) will also provide some information (but likely are less informationally dense). Finally, information about all other products is expected to contain the least information density (*Class P4h*). By constraining the classes so that the less informationally dense classes have lower time delays and contribute fewer additional features, we prevent the inclusion of extraneous, irrelevant features.

Next, we will show how the time lagged data is constructed to form the augmented features set. An expansion of the feature set is referred to as a *lag scheme expansion*. Of course, the optimal lag scheme may be different for each variable modeled; a search of the possible configurations is performed to find the best performing configuration in each prediction class.

The number of possible lag schemes as formulated with the hierarchies in Figures 7.4 and 7.5 for a maximum time delay of 16 days are 11172^4 for both products and components. Without

[3]This symbol refers to the lack of any observation.

[4]This is the subset of the 117649 configurations where all constraints between classes are satisfied. For example, if in a specific configuration class P2 contains time delays $\{0, 1, 2\}$ it would not be permissible for P3 to have time delays $\{0, 1, 2, 4\}$. All configurations with this set of values would be discarded from the set of possible lag configurations.

Class	Lagged Offsets									
	0	1	2	3	4	5	6	7	8	9
P3	•	•	•	•	•	•	•	•	•	•
P3b	•									
P3c	•									
P2	•	•	•	•	•					
P4	•									
P4h	•									

(a)

Class	Lagged Offsets					
	0	1	2	4	8	16
P3	•	•	•	•	•	
P3b	•	•	•	•	•	
P3c						
P2	•	•	•	•	•	
P4	•					
P4h						

(b)

Class	Lagged Offsets									
	0	1	2	3	4	5	6	7	8	9
C4	•	•	•	•	•	•	•	•		
C4b										
C4h										
C3										
C3c										
C2										

(c)

Class	Lagged Offsets					
	0	1	2	4	8	16
C4	•	•	•	•	•	
C4b						
C4h						
C3	•	•	•	•	•	
C3c						
C2	•	•	•	•	•	

(d)

Table 7.3: The optimal lag schemes for the the four prediction classes: (a) current product prediction (expands to 57 augmented features), (b) future product prediction (51 augmented features), (c) current component prediction (10 augmented features), and (d) future component prediction (102 augmented features).

the constraints between classes, there are 117649 configurations[5] of the 6 feature classes if constrained to possible time delays of $\{\varnothing, 0, 1, 2, 4, 8, 16\}$, but many of these configurations are uninteresting variants. Finally, without the hierarchical segmentation and constraints between classes, there are 7^{101} ($\approx 10^{85}$) configurations of the 101 original features. Thus imposing both the feature classification and the constraint hierarchy allows for a greater range of "interesting" lag schemes to be tested for the same amount of lag scheme search.

The optimal lag schemes we use in our experimental results in each of the four prediction classes are provided in Table 7.3. For instance, the current product prediction shown in Table 7.3(a) uses 35 raw features that are expanded into 56 augmented features. Feature classes contributing features are: P3 (1 raw feature $\times$ 10 discrete lags), P3b (5 raw features (P3 already contains the 6th feature) $\times$ 1 discrete lag), P3c (10 raw features (P3 and P3b already handle 6 of these) $\times$ 1 discrete lag), P2 (3 raw features $\times$ 5 discrete lags), P4 (7 raw features $\times$ 1 discrete lag), P4h (9 raw features $\times$ 1 discrete lag), and two game invariant features (game storage cost, game interest rate). While a domain expert could conceive of a generally high-performance feature set, the automated lag scheme search produces a configuration similar to what a domain expert could build without the cost of requiring a domain expert. Also, the results of the optimal lag scheme search can elicit some surprising relationships found in the data.

[5]There are 7 possible configurations of each class, and there are 6 classes. Therefore, there are $7^6 = 117649$ possible configurations.

Reduction of Number of Parameters of the Model. The PLS regression algorithm in [Wold et al., 1983] allows model developers to adjust the model complexity by selecting the number of PLS factors to generate when training. (These factors are analogous to the principal component vectors used in principal component regression.) The number of PLS factors determines the dimensionality of the intermediate variable space that the data is mapped to. The computational complexity does not significantly increase for a larger number of factors but the choice does have an effect on prediction performance: too large a number can cause over-fitting, and too small a number can cause the model to be unable to represent the relationships in the data. The number of factors was systematically varied in the optimal lag scheme search. The best performing number of components is shown for each prediction category in Section 7.1.3.

When computing the optimal lag scheme it is also critical to determine the correct value for model complexity in PLS. Figure 7.6 shows how the prediction accuracy varies based on model complexity relative to the best observed prediction error (the lag scheme is not varied in the data in this graph). The future component class has a slightly different pattern: we conjecture that future component achieves optimal error with a lower number of latent variables because it has a relatively larger number of inputs from the lag scheme search. Empirical results show that it is generally better to err on the side of excess model complexity but excess complexity can also reduce prediction accuracy by, in one case, over 10% above the lowest achievable error.

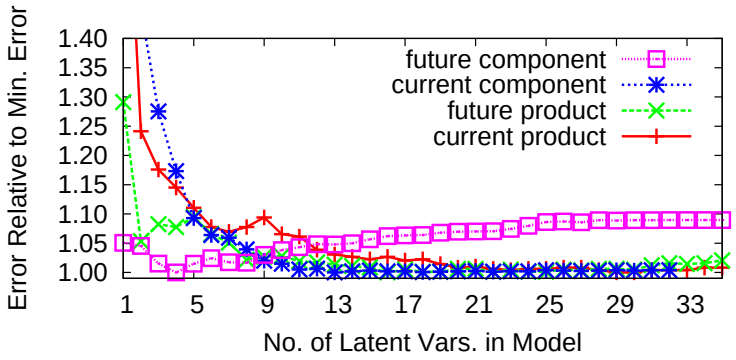


Figure 7.6: Effect of varying the model complexity (number of PLS factors) for the optimal lag scheme. The number of factors used in our experiments varied by prediction class: (1) current product used 18 factors, (2) future product used 22 factors, (3) current component used 22 factors, and (4) future component used 4 factors.

7.1.3 Experimental Results

We evaluate our approach on the TAC SCM 2008 Prediction challenge dataset consisting of 48 experimental runs divided into 3 sets of 16 games. Each set has a different mixture of agents.

The games were divided using a standard 6-fold cross validation for training and scoring. For example, the first fold consisted of a training set of games 9-16 in set A and all games from sets B and C, and a test set of games 1-8 in set A.

	Current Product	Future Product (p-val)	Current Component	Future Component (p-val)
DGFS Exh. (geom. lags)	0.06308	**0.08553** (0.2660)	0.04063	**0.09910** (0.0012)
DGFS Exh. (int. lags)	0.06295	0.09264	0.04008	0.10020
DGFS Exh. (no lags)	0.06300	0.09039	0.04468	0.10947
2008 First Place Agent	0.04779	0.08726	0.03476	0.09964
2008 Third Place Agent	0.0531	0.09934	0.04029	0.10281

Table 7.4: Comparison between the error scores of an agent implementing our method for each of the 4 classes of prediction and the scores of the top performing agents in the 2008 Prediction Challenge. Values in parenthesis are the Student's t-test p-value when compared against 2008 First Place agent. A t-test is only provided for our algorithm when it performs better than the 2008 prediction challenge first-place results. A p-value (p-val) of less than 0.05 indicates that the results of the two algorithms being compared are statistically different at the 95% confidence level.

In order to compute prices for all variables in each prediction class, one prediction model is generated for each variable to predict. The number of prediction models by class is:

1. 16 for current product prices (one for each product type),
2. 16 for future product prices,
3. 16×59 for current component prices (one for each component line-lead time pair for lead times 2 to 60), and
4. 16×26 for future component prices (lead times 5 to 30)

The lag scheme used for each prediction class was computed using an exhaustive search over all lag schemes with valid hierarchical relationships on folds 1, 3, and 5 of the 6-fold cross validation set. The results shown in Table 7.4 are the aggregate scores obtained from experiments on the entire 6-fold set using the best performing lag schemes. The results show that incorporating time-delayed features onto the prediction task does improve the overall prediction accuracy. A comparison with the published results shows that our method is competitive against state-of-the-art methods used within the top performing agents. In particular, our method outperforms existing methods on 20 day ahead predictions in both the component market and the product market. The prediction category for which our method does worst ("current product") can be explained by the fact that the regression model computes a point estimate for sales of each product on each day.There may be some attribute in product requests that causes prices to vary significantly among the set of requests for the same product on the same day. For "future

product," a point estimate is sufficient because the true value represents the median price for product sales 20 days ahead.

The optimal lag schemes used in each of the four prediction classes is provided in Table 7.3. There are some feature classes that are not included at all in the optimal lag scheme, an example of this appears in Table 7.3(c): lagged data from class C4 are the only features used in the prediction model. This suggests that the other variables are not significant to the prediction of current component prices.

7.1.4 Conclusions on TAC SCM Price Prediction

We have shown a prediction method which (1) includes time-delayed observations as additional elements in the feature vector, (2) segments the features hierarchically, and (3) prunes the classes of features used to find the optimal combination of features. Despite the fact our method uses very little domain-knowledge, we achieved competitive (and often superior) performance compared to the state-of-the-art domain-specific prediction techniques in the 2008 TAC SCM Prediction Challenge competition. This general approach also gives an additional benefit of showing the most relevant features by considering the lag schemes chosen in the search.

Chapter 8

Conclusion and Discussion

This thesis addresses the challenges of feature selection when building regression models on real-world data involving many available variables and relatively little training data. While fully automated machine learning and feature selection methods can build accurate prediction models when the dataset is large or unlimited, often the results on small datasets are unsatisfactory. To improve model accuracy in these settings, we develop a novel way of constructing a feature set that incorporates some domain knowledge in the form of a hierarchy, and we present several methods to decide which features to include in the prediction set and which ones to prune based on the hierarchy. We compare exploration of the feature set space using exhaustive as well as several types of stochastic search (greedy, random, or heuristic). Overall, this speeds up the search for an accurate and minimal feature set that leverages the model developer's understanding of the domain as well. limited.

Overall, the domain independent contributions are:

1. a novel method of including multiple temporal data sources as well as spatial information to improve prediction performance and

2. the automated discovery of temporal relationships between data streams to facilitate domain understanding.

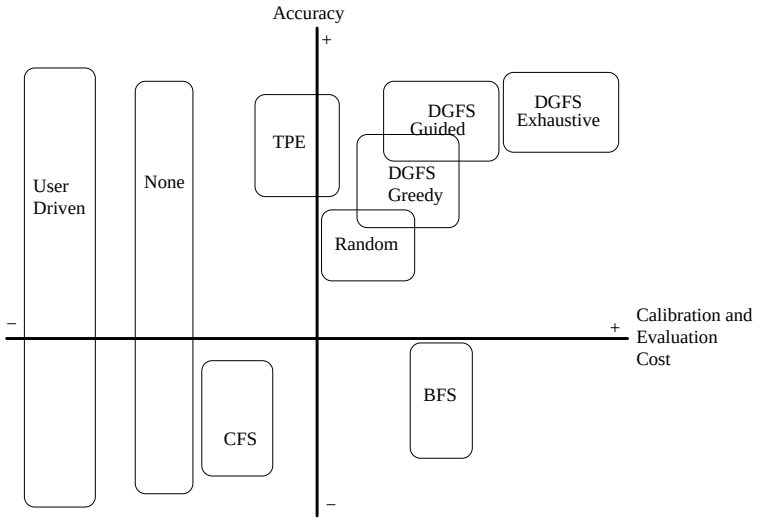
From an intellectual point of view, this work serves to remind practitioners that prediction performance is often lost when completely domain agnostic approaches are employed for several reasons. First, the underlying generator may not be in the model space (model choice error), the parameter values may be changing over time (concept drift), and, the training data may be insufficient to fully calibrate the model due to noise (calibration error). Domain-specific models often minimize these types of error. One extreme is highly optimized domain-specific modeling

such as those employed in TAC SCM where the underlying price generation mechanism is known but the inputs to the mechanism cannot be observed directly. Some agents model these latent unobservable variables and use the underlying price discovery mechanism to make predictions. This approach is highly domain dependent and has no direct application in a domain with a different price finding mechanism. The other extreme includes situations where no domain knowledge is used in machine learning model or feature selection. Fully automated techniques can produce good prediction results when the amount of calibration data is sufficient and the concept being learned is simple. For example, spam detection is such a domain. However, in situations where training data is limited, biasing the model with domain knowledge is beneficial, as we have shown in this thesis.

This work proposes a middle ground between these two extremes where a novice practitioner's domain knowledge can be used to bias the model selection for improved results. DGFS is positioned in this middle ground on the spectrum of feature selection approaches shown in Figure 2.1.

A qualitative way of comparing feature selection approaches is by considering computational effort and overall model accuracy. Figure 8.1 shows the relative computational cost of various feature selection approaches on the horizontal axis along with the relative accuracy on the vertical axis. Algorithms that are further toward the upper right of the figure are more "efficient." Algorithm performance can vary dramatically based on the domain, the dataset and the training set size; the size of the rectangle for each algorithm is drawn reflect this variance.

Figure 8.1: Feature selection computational effort and model accuracy plot. TPE is the Tree of Parzen Estimator method for parameter estimation of Bergstra and Bengio [2012]. None is the no feature selection approach.



Choosing the most appropriate feature selection algorithm for a particular application should account for these differences in human and computational effort. DGFS is most suitable in cases where additional accuracy is desired on datasets with relatively few training examples and where additional human effort in modeling is available. This describes many real-world instances of prediction.

DGFS has an additional benefit of providing domain understanding. By considering relationships of the features included in the lag scheme configuration, users can have greater assurance that the relevant relationships in the domain are incorporated. Also, the lag schemes can reveal novel relationships that domain practitioners may not have considered previously. This is particularly evident in the airline ticket price prediction models of Chapter 5 which show different lag scheme configurations for different routes and user preferences.

8.1 Future Directions for Research

As outlined in the preceding chapters, there are areas in which this line of research can be further expanded. The following are several possible future directions.

Model analysis and visualization. We believe that automated analysis and visualization of the machine learning model (including the pruned feature subset and the calibrated machine learning model) can facilitate better domain understanding for users. Putting greater knowledge of the prediction model in the hands of the users can enable greater confidence in the models developers actually apply. As a first step in this direction, we show the lag scheme configurations and learning model coefficient weights from the experiments in the preceding chapters. However, we believe there is further value in providing the structure of the calibrated machine learning models to the user. This level of knowledge may be of interest to the user for both a sanity check on the generated models and in facilitating domain understanding. We consider this kind of analysis as a significant and generally underexplored area of applied machine learning.

Hyperparameter optimization for temporal features. This thesis has considered the problem of feature selection using several general search approaches: exhaustive, greedy, and heuristic ("guided"). Using the state of the art in hyper-parameter optimization, we have improved the search efficiency of search for feature selection. However, we believe there is still space for further improvements in search efficiency by explicitly modeling the temporal relationships between features. We leave this as an area of future work.

Automated construction of the feature classes and relationships. In the domains we analyze, the domain knowledge (in the form of feature classes and constraints) is provided by the model developer. It may also be possible to build these structures in an automated way.

This would allow us to come full circle and provide a fully automated version of DGFS which builds this information as intermediate intermediate data structure in an automated learning process.

References

Abdi, H. (2010). Partial least squares regression and projection on latent structure regression (pls regression). *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):97–106.

Abu-Mostafa, Y. S. (1989). The vapnik-chervonenkis dimension: Information versus complexity in learning. *Neural Computation*, 1(3):312–317.

Agrawal, R., Ieong, S., and Velu, R. (2011). Timing when to buy. In *Proc. of the 20th ACM Int'l Conference on Information and Knowledge Management*, pages 709–718, New York, NY, USA. ACM.

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723.

Bachis, E. and Piga, C. (2006). Hub premium, airport dominance and market power in the european airline industry. *Rivista di Politica Economica*, 96(5):11–54.

Bachis, E. and Piga, C. (2007). On-line price discrimination with and without arbitrage conditions. In *Loughborough University, Department of Economics Working Paper. Presented at the International Industrial Organization Conference, Savannah, GA*.

Bachis, E. and Piga, C. A. (2011). Low-cost airlines and online price dispersion. *Int'l Journal of Industrial Organization*, 29(6):655–667.

Bagheri, M. A., Gao, Q., and Escalera, S. (2012). Efficient pairwise classification using local cross off strategy. In *Advances in Artificial Intelligence*, pages 25–36. Springer.

Bajari, P. and Hortacsu, A. (2003). The winner's curse, reserve prices, and endogenous entry: empirical insights from ebay auctions. *RAND Journal of Economics*, 34(2):329–355.

Barbosa, R. P. and Belo, O. (2008). Autonomous forex trading agents. In *IEEE Int'l Conf. on Data Mining (ICDM)*, pages 389–403.

Belobaba, P. P. (1987). Airline yield management. An overview of seat inventory control. *Transportation Science*, 21(2):63–73.

Benisch, M., Greenwald, A., Grypari, I., Lederman, R., Naroditskiy, V., and Tschantz, M. (2004). Botticelli: A supply chain management agent designed to optimize under uncertainty. *ACM Trans. on Comp. Logic*, 4(3):29–37.

Benisch, M., Sardinha, A., Andrews, J., Ravichandran, R., and Sadeh, N. (2009). Cmieux: Adaptive strategies for competitive supply chain trading. *Electronic Commerce Research and Applications*, 8(2):78 – 90.

Benisch, M., Sardinha, A., Andrews, J., and Sadeh, N. (2006). CMieux: adaptive strategies for competitive supply chain trading. In *Proc. of 8th Int'l Conf. on Electronic Commerce*, pages 47–58. ACM Press.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554.

Bing! Travel (2012). Website. `http://www.bing.com/travel/`.

Box, G., Pelham, E., and Jenkins, G. M. (1994). *Time Series Analysis: Forecasting and Control*. Prentice Hall PTR, 3rd edition.

Box, G. E., Jenkins, G. M., and Reinsel, G. C. (2013). *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. John Wiley & Sons, Hoboken, New Jersey.

Cardie, C. (1993). Using decision trees to improve case-based learning. In *International Conference in Machine Learning (ICML)*, volume 93, pages 25–32.

Collins, J., Arunachalam, R., Sadeh, N., Ericsson, J., Finne, N., and Janson, S. (2006). The supply chain management game for the 2006 trading agent competition. Technical Report CMU-ISRI-07-100, Carnegie Mellon University, Pittsburgh, PA.

Collins, J., Ketter, W., and Gini, M. (2009). Flexible decision control in an autonomous trading agent. *Electronic Commerce Research and Applications*, 8(2):91–105.

Contreras, J., Espinola, R., Nogales, F., and Conejo, A. (2003). Arima models to predict next-day electricity prices. *IEEE Transactions on Power Systems*, 18(3):1014–1020.

Cun, Y. L., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in Neural Information Processing Systems 2*, pages 598–605, San Francisco, CA. Morgan Kaufmann.

Dayal, B. S. and MacGregor, J. F. (1997). Recursive exponentially weighted pls and its applications to adaptive control and prediction. *Journal of Process Control*, 7(3):169–179.

de Jong, S. (1993). Simpls: An alternative approach to partial least squares regression. *Chemometrics and Intelligent Laboratory Systems*, 18(3):251–263.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30.

Dietterich, T. G. (2002). Machine learning for sequential data: A review. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 15–30. Springer.

Du, K.-L. and Swamy, M. (2014). Recurrent neural networks. In *Neural Networks and Statistical Learning*, pages 337–353. Springer, London, UK.

Ebay Inc (2012). Website. `http://www.ebay.com/`.

Elmaghraby, W. and Keskinocak, P. (2003). Dynamic pricing in the presence of inventory considerations: research overview, current practices, and future directions. *Management Science*, 49(10):1287–1309.

Ergon, R. (2002). Pls score–loading correspondence and a bi-orthogonal factorization. *Journal of Chemometrics*, 16(7):368–373.

Etzioni, O., Tuchinda, R., Knoblock, C., and Yates, A. (2003). To buy or not to buy: mining airfare data to minimize ticket purchase price. In *SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 119–128, New York, NY, USA. ACM.

Fahlman, S. E. and Lebiere, C. (1990). The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pages 524–532, San Francisco, CA. Morgan Kaufmann.

Fayyad, U. and Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, Chambery, France. AAAI.

Francis, G., Humphreys, I., Ison, S., and Aicken, M. (2006). Where next for low cost airlines? a spatial and temporal comparative study. *Journal of Transport Geography*, 14(2):83–94.

Gama, J. and Gaber, M. M. (2007). *Learning from Data Streams: Processing Techniques in Sensor Networks*. New generation computing. Springer.

Giacomini, R. and Granger, C. W. J. (2004). Aggregation of space-time processes. *Journal of Econometrics*, 118(1-2):7–26.

Gigerenzer, G. (2004). Mindless statistics. *The Journal of Socio-Economics*, 33(5):587–606.

Groves, W. (2013). Using domain knowledge to systematically guide feature selection. In *Proc. Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, pages 3215–3216, Palo Alto, CA. AAAI Press.

Groves, W., Collins, J., and Gini, M. (2009). Visualization and analysis methods for comparing agent behavior in TAC SCM. In *Proc. 8th Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, Budapest, Hungary.

Groves, W., Collins, J., Gini, M., and Ketter, W. (2014). Agent-assisted supply chain management: Analysis and lessons learned. *Journal of Decision Support Systems*, 57:274–284.

Groves, W. and Gini, M. (2013a). Improving prediction in tac scm by integrating multivariate and temporal aspects via pls regression. In *Agent-Mediated Electronic Commerce. Designing Trading Strategies and Mechanisms for Electronic Markets*, volume 119 of *Lecture Notes in Business Information Processing*, pages 28–43. Springer, Berlin.

Groves, W. and Gini, M. (2013b). Optimal airline ticket purchasing using automated user-guided feature selection. In *Proc. Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, pages 150–156, Palo Alto, CA. AAAI Press.

Groves, W. and Gini, M. (2015). On optimizing airline ticket purchasing. *ACM Transactions on Intelligent Systems and Technology (TIST)*, accepted, in press.

Groves, W., Ketter, W., Collins, J., and Gini, M. (2010). Analyzing market interactions in a multi-agent supply chain environment. In Sharman, R., Rao, H. R., and Raghu, T. S., editors, *Exploring the Grand Challenges for Next Generation E-Business. 8th Workshop on E-Business, WEB 2009, Revised Selected Papers*, volume 52 of *Lecture Notes in Business Information Processing*, pages 44–58. Springer.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182.

Hall, M. A. (1999). *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato.

Hall, M. A. (2000). Correlation-based feature selection for discrete and numeric class machine learning. In *Int'l Conf. on Machine Learning (ICML)*, pages 359–366, San Francisco, CA. Morgan Kaufmann.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, New York, NY, USA. Chapter 7.

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized additive models*, volume 43. CRC Press.

Hillmer, S. C., Larcker, D. F., and Schroeder, D. A. (1983). Forecasting accounting data: A multiple time-series analysis. *Journal of Forecasting*, 2(4):389–404.

Hoerl, A. E. and Kennard, R. W. (2000). Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, 42(1):80–86.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In Coello, C. A., editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer Berlin Heidelberg.

Imrie, C. E., Durucan, S., and Korre, A. (2000). River flow prediction using artificial neural networks: generalisation beyond the calibration range. *Journal of Hydrology*, 233(1-4):138–153.

John, G. H., Kohavi, R., Pfleger, K., et al. (1994). Irrelevant features and the subset selection problem. In *International Conference of Machine Learning (ICML)*, volume 94, pages 121–129.

Johnson, D. H. (1999). The insignificance of statistical significance testing. *The Journal of Wildlife Management*, pages 763–772.

Jolliffe, I. T. (1982). A note on the use of principal components in regression. *J. of Royal Statistical Society. (Applied Statistics)*, 31(3):300–303.

Kaufman, P. (2005). *New Trading Systems and Methods*. John Wiley & Sons, Hoboken, New Jersey.

Kearns, M. and Ortiz, L. (2003). The Penn-Lehman Automated Trading Project. *IEEE Intelligent Systems*, 18(6):22–31.

Kern, J., Characklis, G., Doyle, M., Blumsack, S., and Whisnant, R. (2011). Influence of deregulated electricity markets on hydropower generation and downstream flow regime. *Journal of Water Resources Planning and Management*, 1:134.

Ketter, W., Collins, J., Gini, M., Gupta, A., and Schrater, P. (2009). Detecting and Forecasting Economic Regimes in Multi-Agent Automated Exchanges. *Decision Support Systems*, 47(4):307–318.

Kiekintveld, C., Miller, J., Jordan, P. R., Callender, L. F., and Wellman, M. P. (2009). Forecasting market prices in a supply chain game. *Electronic Commerce Research and Applications*, 8(2):63 – 77.

Kiekintveld, C., Miller, J., Jordan, P. R., and Wellman, M. P. (2006). Controlling a supply chain agent using value-based decomposition. In Feigenbaum, J., Chuang, J. C.-I., and Pennock, D. M., editors, *ACM Conference on Electronic Commerce*, pages 208–217. ACM.

Kiekintveld, C., Miller, J., Jordan, P. R., and Wellman, M. P. (2007). Forecasting market prices in a supply chain game. In Durfee, E. H., Yokoo, M., Huhns, M. N., and Shehory, O., editors, *AAMAS*, page 234. IFAAMAS.

Kira, K. and Rendell, L. A. (1992). The feature selection problem: Traditional methods and a new algorithm. In *Proc. of the Tenth Nat'l Conf. on Artificial Intelligence (AAAI)*, pages 129–134.

Kitanidis, P. and Bras, R. (1980). Real-time forecasting with a conceptual hydrologic model 1. analysis of uncertainty. *Water Resources Research*, 16(6):1025–1033.

Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324.

Kononenko, I. (1994). Estimating attributes: analysis and extensions of relief. In *European Conference in Machine Learning (ECML)*, pages 171–182. Springer.

Kugiumtzis, D. (1996). State space reconstruction parameters in the analysis of chaotic time series — the role of the time window length. *Physica D: Nonlinear Phenomena*, 95(1):13–28.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 473–480, New York, NY, USA. ACM.

Leahy, P., Kiely, G., and Corcoran, G. (2008). Structural optimisation and input selection of an artificial neural network for river level prediction. *Journal of Hydrology*, 355(1-4):192–201.

Levin, Y., McGill, J., and Nediak, M. (2009). Dynamic pricing in the presence of strategic consumers and oligopolistic competition. *Management Science*, 55(1):32–46.

Li, Z., Dunham, M., and Xiao, Y. (2003). Stiff: A forecasting framework for spatiotemporal data. In *Mining Multimedia and Complex Data*, volume 2797 of *Lecture Notes in Computer Science*, pages 183–198. Springer Berlin / Heidelberg.

Lin, M., Lucas Jr, H. C., and Shmueli, G. (2013). Research commentary-too big to fail: Large samples and the p-value problem. *Information Systems Research*, 24(4):906–917.

Loh, W.-Y. (2011). Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23.

Lucking-Reiley, D., Bryan, D., Prasad, N., and Reeves, D. (2007). Pennies from Ebay: the determinants of price in online auctions. *The Journal of Industrial Economics*, 55(2):223–233.

Mantin, B. and Gillen, D. (2011). The hidden information content of price movements. *European Journal of Operational Research*, 211(2):385–393.

Mantin, B. and Koo, B. (2010). Weekend effect in airfare pricing. *Journal of Air Transport Management*, 16(1):48–50.

Martens, H. and Næs, T. (1992). *Multivariate Calibration*. John Wiley & Sons, Hoboken, New Jersey.

Molina, L. C., Belanche, L., and Nebot, À. (2002). Feature selection algorithms: a survey and experimental evaluation. In *IEEE Int'l Conf. on Data Mining (ICDM)*, pages 306–313, Piscataway, NJ. IEEE.

Obeng, K. and Sakano, R. (2012). Airline fare and seat management strategies with demand dependency. *Journal of Air Transport Management*, 24(0):42–48.

Olason, T. and Watt, W. E. (1986). Multivariate transfer function-noise model of river flow for hydropower operation. *Nordic Hydrology*, 17(3):185–202.

Pace, R. K., Barry, R., Clapp, J. M., and Rodriquez, M. (1998). Spatiotemporal autoregressive models of neighborhood effects. *The Journal of Real Estate Finance and Economics*, 17(1):15–33.

Pace, R. K., Barry, R., Gilley, O. W., and Sirmans, C. F. (2000). A method for spatial-temporal forecasting with an application to real estate prices. *International Journal of Forecasting*, 16(2):229 – 246.

Pardoe, D. and Stone, P. (2007). An autonomous agent for supply chain management. In Adomavicius, G. and Gupta, A., editors, *Handbooks in Information Systems Series: Business Computing*. Elsevier.

Pardoe, D. and Stone, P. (2008). The 2007 TAC SCM prediction challenge. In *AAAI 2008 Workshop on Trading Agent Design and Analysis*.

Parzen, E. (1962). On estimation of a probability density function and mode. *The annals of mathematical statistics*, pages 1065–1076.

Piga, C. A. and Filippi, N. (2002). Booking and flying with low-cost airlines. *Int'l Journal of Tourism Research*, 4(3):237–249.

Puller, S. L. and Taylor, L. M. (2012). Price discrimination by day-of-week of purchase: Evidence from the u.s. airline industry. *Journal of Economic Behavior & Organization*, 84(3):801–812.

Quinlan, J. and Cameron-Jones, R. (1995). Oversearching and layered search in empirical learning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1019–1024. Morgan Kaufmann.

Rasmussen, C. E. (2006). *Gaussian processes for machine learning*. MIT Press.

Raykhel, I. and Ventura, D. (2009). Real-time automatic price prediction for eBay online trading. In *Proc. Innovative Applications of Artificial Intelligence Conf.*, pages 135–140, Palo Alto, CA. AAAI.

Rhoades, S. A. (1993). Herfindahl-hirschman index, the. *Federal Reserve Bulletin*, 79:188–189.

Rodrigues, P., Gama, J., and Pedroso, J. (2008). Hierarchical clustering of time-series data streams. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):615–627.

Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*, volume 2. Prentice Hall, Englewood Cliffs.

Sadeh, N., Arunachalam, R., Eriksson, J., Finne, N., and Janson, S. (2003). TAC-03–A Supply-Chain Trading Competition. *AI magazine*, 24(1):92.

Sauer, T. (1994). Time series prediction by using delay coordinate embedding. In Weigend, A. S. and Gershenfeld, N. A., editors, *Time series prediction: forecasting the future and understanding the past*, pages 175–194. Addison Wesley, Boston, MA.

Schölkopf, B., Smola, A. J., Williamson, R. C., and Bartlett, P. L. (2000). New support vector algorithms. *Neural computation*, 12(5):1207–1245.

Sexton, R., Dorsey, R., and Sikander, N. (2004). Simultaneous optimization of neural network function and architecture algorithm. *Decision Support Systems*, 36(3):283–296.

Shekhar, S. and Chawla, S. (2003). *Spatial Databases: a Tour*, volume 2003. Prentice Hall, Upper Saddle River, NJ.

Shekhar, S., Evans, M., Kang, J., and Mohan, P. (2011). Identifying patterns in spatial information: a survey of methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):193–214.

Sherstov, A. A. and Stone, P. (2004). Three automated stock-trading agents: A comparative study. In Faratin, P. and Rodríguez-Aguilar, J. A., editors, *AMEC*, volume 3435 of *Lecture Notes in Computer Science*, pages 173–187. Springer.

Singh, V. (2013). *Entropy Theory and its Application in Environmental and Water Engineering*. Wiley.

Smith, B. C., Leimkuhler, J. F., and Darrow, R. M. (1992). Yield management at American Airlines. *Interfaces*, 22(1):8–31.

Stan, M., Stan, B., and Florea, A. M. (2006). A dynamic strategy agent for supply chain management. In *SYNASC*, pages 227–232. IEEE Computer Society.

Stone, P. and Greenwald, A. R. (2005). The first international trading agent competition: Autonomous bidding agents. *Electronic Commerce Research*, 5(2):229–265.

Subramanian, J., Stidham, Shaler, J., and Lautenbacher, C. J. (1999). Airline yield management with overbooking, cancellations, and no-shows. *Transportation Science*, 33(2):147–167.

Tiao, G. C. and Box, G. E. P. (1981). Modeling multiple time series with applications. *Journal of the American Statistical Association*, 76(376):802–816.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

U.S. Department of Transportation (2012). Origin-destination survey. `http://www.bts.gov/`. Bureau of Transportation Services.

Vapnik, V. N. (1998). *Statistical learning theory*, volume 2. Wiley, New York.

Vowles, T. M. (2000). The effect of low fare air carriers on airfares in the us. *Journal of Transport Geography*, 8(2):121 – 128.

Wan, E. A. (1994). Time series prediction by using a connectionist network with internal delay lines. In Weigend, A. S. and Gershenfeld, N. A., editors, *Time series prediction: forecasting the future and understanding the past*, pages 195–218, Boston, MA. Addison Wesley.

Weigend, A. S., Rumelhart, D. E., and Huberman, B. A. (1990). Generalization by weight-elimination with application to forecasting. In Lippmann, R., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 2*, pages 875–882, San Francisco, CA. Morgan Kaufmann.

Wellman, M. P., Reeves, D. M., Lochner, K. M., and Vorobeychik, Y. (2004). Price prediction in a trading agent competition. *J. Artif. Intell. Res. (JAIR)*, 21:19–36.

Williams, C. K. (1998). Prediction with gaussian processes: From linear regression to linear prediction and beyond. In *Learning in graphical models*, pages 599–621. Springer.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, CA.

Wold, H. O. A. (1966). Nonlinear estimation by iterative least square procedures. *Festschrift for J. Neyman*.

Wold, S., Martens, H., and Wold, H. (1983). Multivariate calibration problem in chemistry solved by the PLS method. *Matrix Pencils*, 973(18):286–293.

Xioufis, E. S., Tsoumakas, G., and Vlahavas, I. P. (2011). Multi-label learning approaches for music instrument recognition. In Kryszkiewicz, M., Rybinski, H., Skowron, A., and Ras, Z. W., editors, *ISMIS*, volume 6804 of *Lecture Notes in Computer Science*, pages 734–743. Springer.

Yang, Y. (2005). Can the strengths of aic and bic be shared? a conflict between model indentification and regression estimation. *Biometrika*, 92(4):937–950.

Zhao, Z. A. and Liu, H. (2011). *Spectral Feature Selection for Data Mining*. Chapman & Hall/CRC, London, UK.

Zhou, Y., Fenton, N., and Neil, M. (2014). Bayesian network approach to multinomial parameter learning using data and expert judgments. *International Journal of Approximate Reasoning*, 55(5):1252 – 1268.

Zimmerman, D. W. and Zumbo, B. D. (1993). Relative power of the wilcoxon test, the friedman test, and repeated-measures anova on ranks. *The Journal of Experimental Education*, 62(1):75–86.

Appendix A

Synthetic Data Experiments (continued from Chapter 4)

This appendix contains detailed statistical outputs from some of the experiments in the rest of this document. This information is provided in an appendix to make the treatments of the experimental results in the preceding chapters more concise.

A graph corresponding to the time series of one of the configuration seeds is shown in Figure A.1.

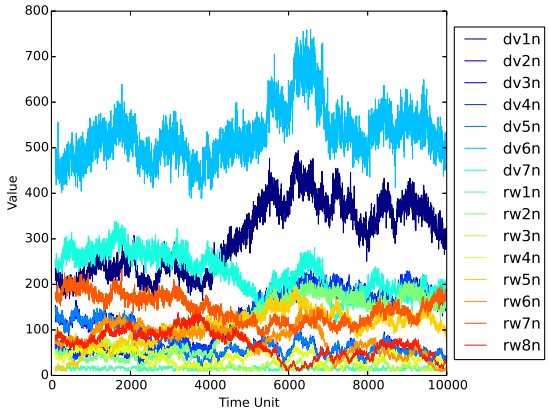


Figure A.1: Time Series for SyntheticRiver dataset showing both obervable and latent variables (seed 0). The simulated units are in kcfs.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Exhaustive	25.5	25.6	25.8	27.1	27.8	28.7	31.1	42.8	42.4	42.0
Greedy	33.5	33.6	33.6	33.9	36.2	38.3	41.1	66.7	85.5	71.2
Random	27.9	27.9	28.2	28.4	30.9	32.3	34.4	51.1	60.7	52.9
Guided	25.5	26.1	26.2	27.2	29.0	30.1	32.5	44.8	54.5	49.3
No FS	32.5	33.9	35.9	38.7	44.3	48.7	64.1	118.0	145.7	121.8
CFSB	27.8	28.0	28.0	31.8	37.1	44.2	70.2	141.5	151.0	461.8
CFSF	25.0	26.4	28.0	28.2	33.4	41.5	65.9	123.4	167.4	201.9
BFSB	23.4	25.1	23.9	27.5	28.9	41.2	57.5	240.9	141.2	976.2
BFSF	91.2	93.7	94.9	99.3	105.9	109.2	113.1	132.2	129.7	126.2

Table A.1: Validation set root-mean-square error for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Exhaustive	3487	2643	1779	1535	1410	1324	1256	1040	1280	1253
Greedy	1	1	0	0	0	0	0	0	0	0
Random	24	16	13	12	10	10	10	10	10	11
Guided	47	38	34	33	32	31	29	32	31	31
No FS	0	0	0	0	0	0	0	0	0	0
CFSB	37	26	18	15	12	11	10	10	9	10
CFSF	34	23	17	13	12	10	9	9	9	9
BFSB	4222	2960	1052	450	284	127	88	63	54	49
BFSF	88	47	33	22	17	16	14	14	12	12

Table A.2: Total feature selection search elapsed-time (rounded to the nearest second) for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Exh.	35273	35273	35273	35273	35273	35273	35273	35273	35273	35273
Greedy	28	28	28	28	28	28	28	28	31	32
Random	307	305	315	307	306	306	311	317	305	309
Guided	374	373	381	377	371	373	387	382	365	363
No FS	1	1	1	1	1	1	1	1	1	1
CFSB	1	1	1	1	1	1	1	1	1	1
CFSF	1	1	1	1	1	1	1	1	1	1
BFSB	1	1	1	1	1	1	1	1	1	1
BFSF	1	1	1	1	1	1	1	1	1	1

Table A.3: Number of feature set evaluations for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

A.1 Incorrect Constraints Experiment

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Greedy	66.4	66.0	68.2	71.6	80.7	83.7	86.8	104.9	72.0	89.0
Guided	26.1	26.5	26.8	28.3	29.7	30.4	34.4	57.3	51.8	55.0
No FS	32.5	33.9	35.9	38.7	44.3	48.7	64.1	118.0	145.7	121.8
Random	28.6	30.3	30.0	33.0	35.3	36.4	41.3	65.4	72.9	60.1

Table A.4: Validation set root-mean-square error for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Greedy	3	2	1	1	1	1	1	1	1	1
Guided	65	54	51	49	48	46	50	44	52	53
No FS	1	0	0	0	0	0	0	0	0	0
Random	23	17	13	12	11	12	12	12	11	11

Table A.5: Total feature selection search elapsed-time for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Greedy	56	56	56	56	56	56	56	56	56	56
Guided	254	270	264	243	253	221	291	326	350	329
No FS	1	1	1	1	1	1	1	1	1	1
Random	163	163	151	159	159	165	163	163	165	153

Table A.6: Number of feature set evaluations for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

A.2 No Constraints Experiment

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
method										
Greedy	33.5	33.6	33.6	33.9	36.2	38.3	41.1	66.7	68.0	59.0
Guided	25.5	25.6	25.8	27.4	29.2	29.0	31.2	45.8	58.0	47.6
No FS	32.5	33.9	35.9	38.7	44.3	48.7	64.1	118.0	145.7	121.8
Random	27.5	27.8	28.1	29.4	29.4	33.1	33.8	52.8	60.6	46.6

Table A.7: No constraints. Validation set root-mean-square error for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
method										
Greedy	3.1	1.9	1.6	1.4	1.1	1.2	1.0	1.1	1.0	1.1
Guided	55.3	45.2	33.7	34.1	37.0	31.8	34.3	33.0	34.9	35.1
No FS	0.6	0.4	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Random	37.3	23.1	19.4	15.4	13.4	12.3	14.7	13.0	13.0	13.8

Table A.8: No constraints. Total feature selection search elapsed-time for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
method										
Greedy	84	84	84	84	84	84	84	92.4	84	84
Guided	400	400	400	400	400	400	400	400.0	400	400
No FS	1	1	1	1	1	1	1	1.0	1	1
Random	400	400	400	400	400	400	400	400.0	400	400

Table A.9: Number of feature set evaluations for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

A.3 Expanded Time Lag Choices

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Greedy	33.5	33.6	33.6	33.9	36.2	38.3	41.1	66.7	85.5	71.2
Guided	26.1	25.7	25.8	28.2	29.2	29.3	33.0	49.3	54.6	47.3
No FS	32.5	33.9	35.9	38.7	44.3	48.7	64.1	118.0	145.7	121.8
Random	28.2	28.8	28.1	29.8	31.4	34.0	35.5	55.3	66.3	49.5

Table A.10: Validation set root-mean-square error for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Greedy	2	2	1	1	1	1	1	1	1	1
Guided	81	69	47	59	52	48	45	52	51	53
No FS	1	1	0	0	0	0	0	0	0	0
Random	37	26	16	16	11	15	13	15	13	16

Table A.11: Total feature selection search elapsed-time (seconds) for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

	1/1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	1/256	1/512
Greedy	48	48	48	48	48	48	48	48	53	48
Guided	347	367	350	357	364	376	364	375	373	362
No FS	1	1	1	1	1	1	1	1	1	1
Random	281	267	269	272	275	279	269	277	270	276

Table A.12: Number of feature set evaluations for various feature selection algorithms coupled with PLS regression applied to SyntheticRiver domain.

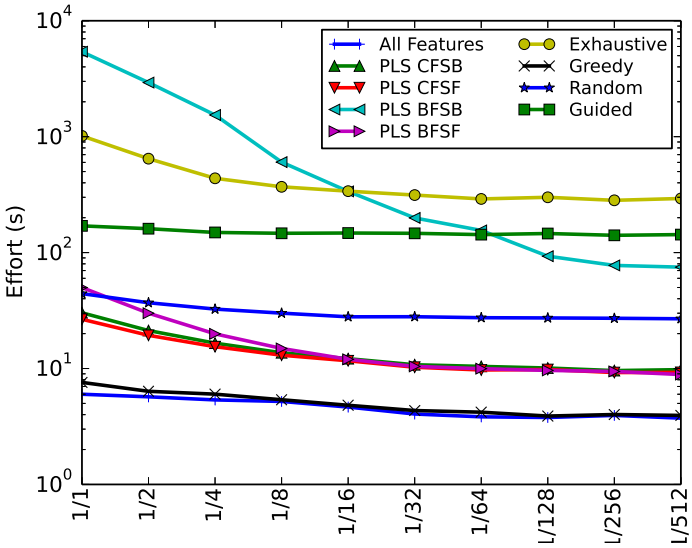


Figure A.2: Computational effort for various methods on SyntheticRiver dataset. Values (in seconds) are presented with a logarithmic y-scale to make differences more visible across the range of error measurements.

A.4 LibSvm

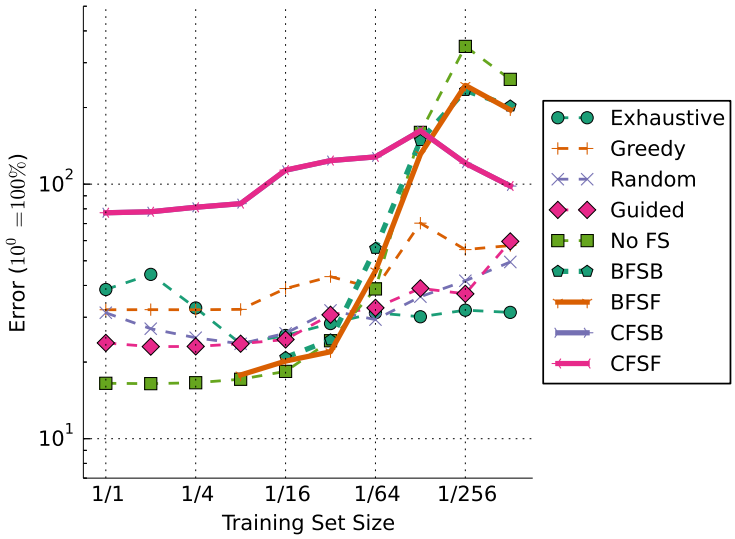


Figure A.3: Efficiency curve for various methods on SyntheticRiver dataset: RMSE versus training set size for various feature selection approaches. Errors are presented with a logarithmic y-scale to make differences more visible across the range of error measurements.

A.5 ElasticNet

The section examines the use of ElasticNet as the underlying machine learning algorithm.

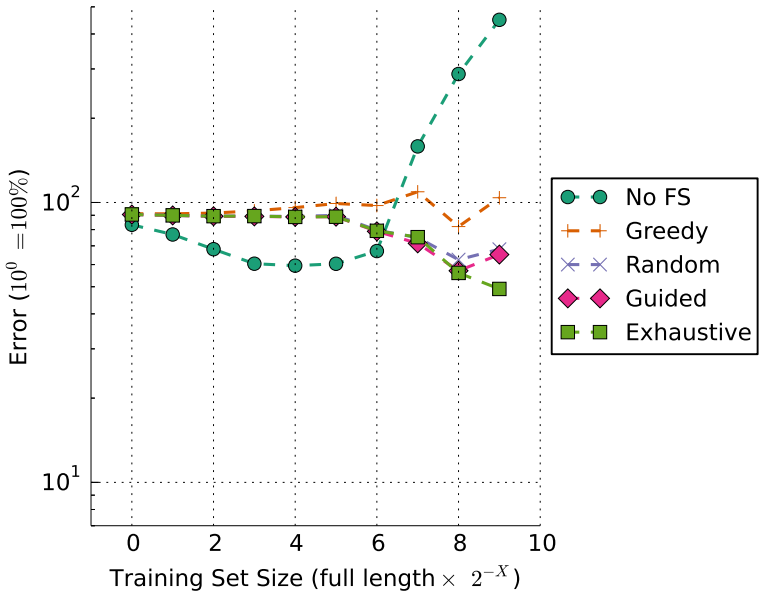


Figure A.4: Efficiency curve for ElasticNet on SyntheticRiver dataset: RMSE versus training set size for various feature selection approaches. Errors are presented with a logarithmic y-scale to make differences more visible across the range of error measurements.

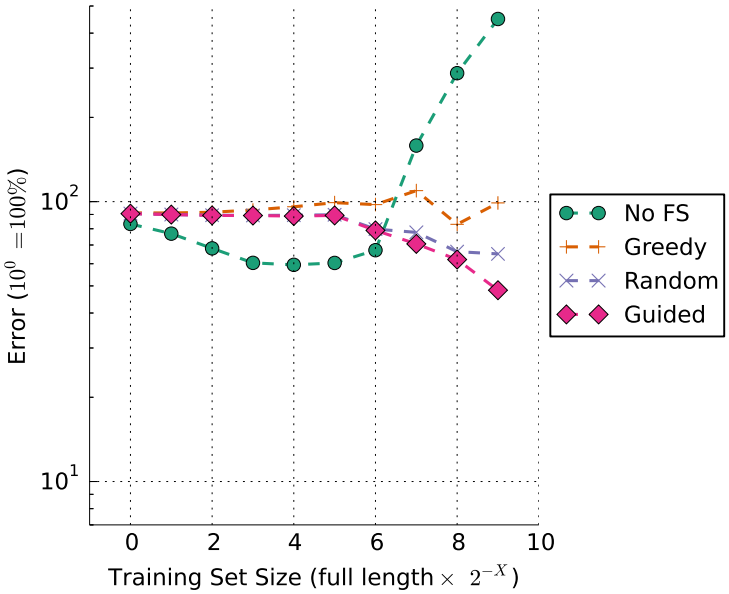


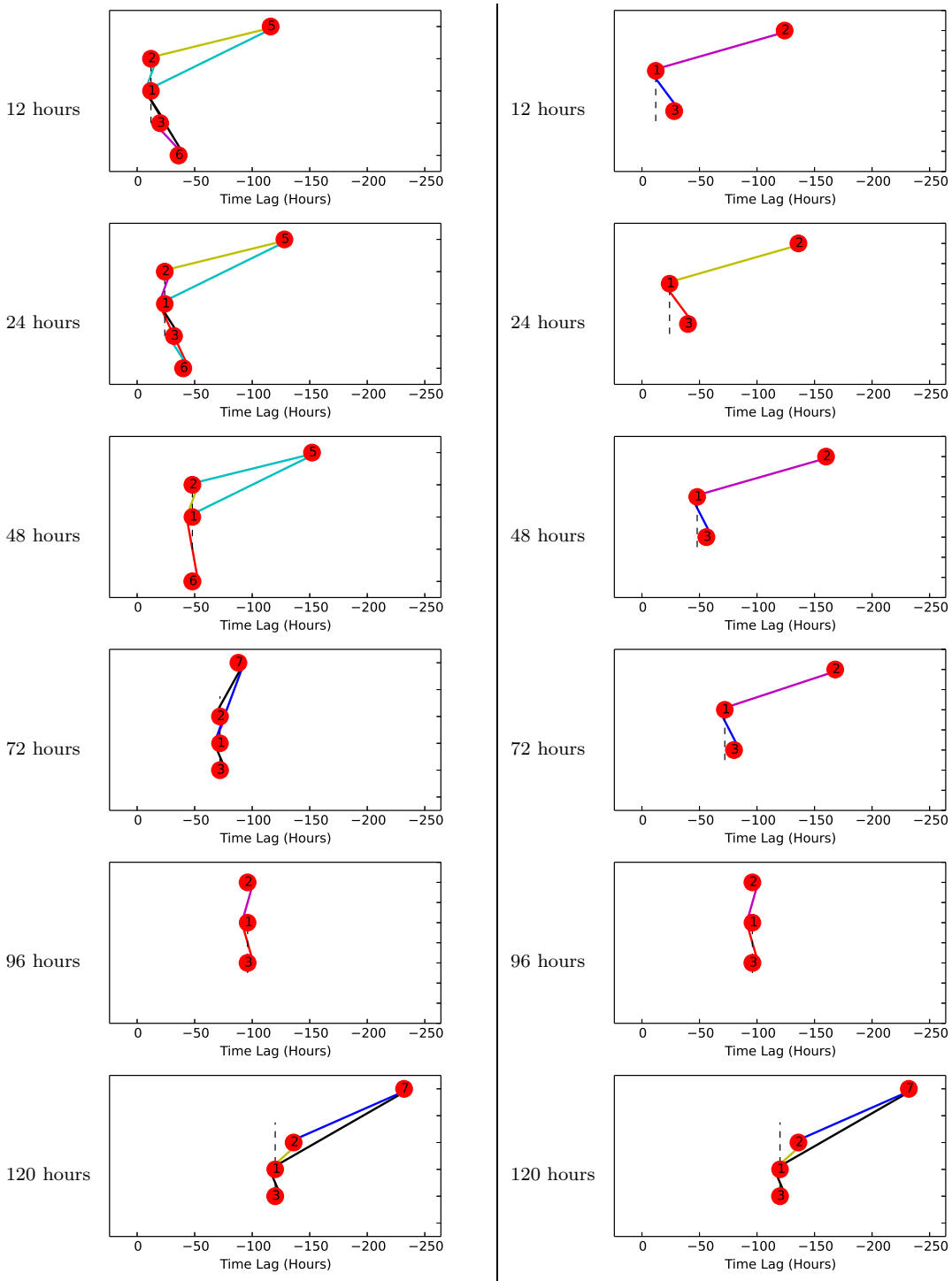
Figure A.5: Efficiency curve for ElasticNet with expanded time lags on SyntheticRiver dataset: RMSE versus training set size for various feature selection approaches. Errors are presented with a logarithmic y-scale to make differences more visible across the range of error measurements.

Appendix B

River Flow Experiments (continued from Chapter 6)

score_{va} score_{te} runtime (secs) # of Vars		**Target Offset**					
Method	**Algorithm**	12	24	48	72	96	120
lastvalue	PLS	12.9 14.6 31.9 1.0	24.1 27.9 28.5 1.0	44.8 52.8 30.1 1.0	63.2 75.5 16.8 1.0	79.1 95.6 14.9 1.0	92.7 112.7 14.9 1.0
lastvalue	Ridge	12.9 14.6 30.2 1.0	24.1 27.9 31.5 1.0	44.8 52.8 31.1 1.0	63.2 75.5 16.7 1.0	79.1 95.6 15.9 1.0	92.7 112.7 20.6 1.0
lastvalue	nuSVR	12.6 14.6 42.5 1.0	23.5 28.0 55.6 1.0	43.5 52.9 201.5 1.0	61.1 75.9 177.6 1.0	76.4 96.4 270.8 1.0	89.1 113.9 110.4 1.0
mostrecent	zeror	201.1 260.6 18.8 6.0	200.8 260.3 17.6 6.0	200.4 259.9 15.5 6.0	200.0 259.7 14.9 6.0	199.7 259.5 15.4 6.0	199.4 259.5 15.3 6.0
uts8h5d	PLS	9.8 9.1 14.4 15.0	18.3 18.2 14.6 15.0	36.6 38.9 15.8 15.0	55.2 62.0 14.6 15.0	72.3 84.4 17.8 15.0	87.0 104.4 15.0 15.0
uts8h5d	Ridge	9.6 9.0 13.9 15.0	18.2 18.0 14.8 15.0	36.5 38.7 15.8 15.0	55.0 61.7 14.3 15.0	72.1 84.2 18.6 15.0	86.9 104.2 15.0 15.0
uts8h5d	nuSVR	11.9 14.3 169.9 15.0	20.9 25.6 115.5 15.0	38.3 46.5 86.8 15.0	55.6 68.5 65.4 15.0	71.3 89.4 82.0 15.0	84.7 107.6 60.0 15.0

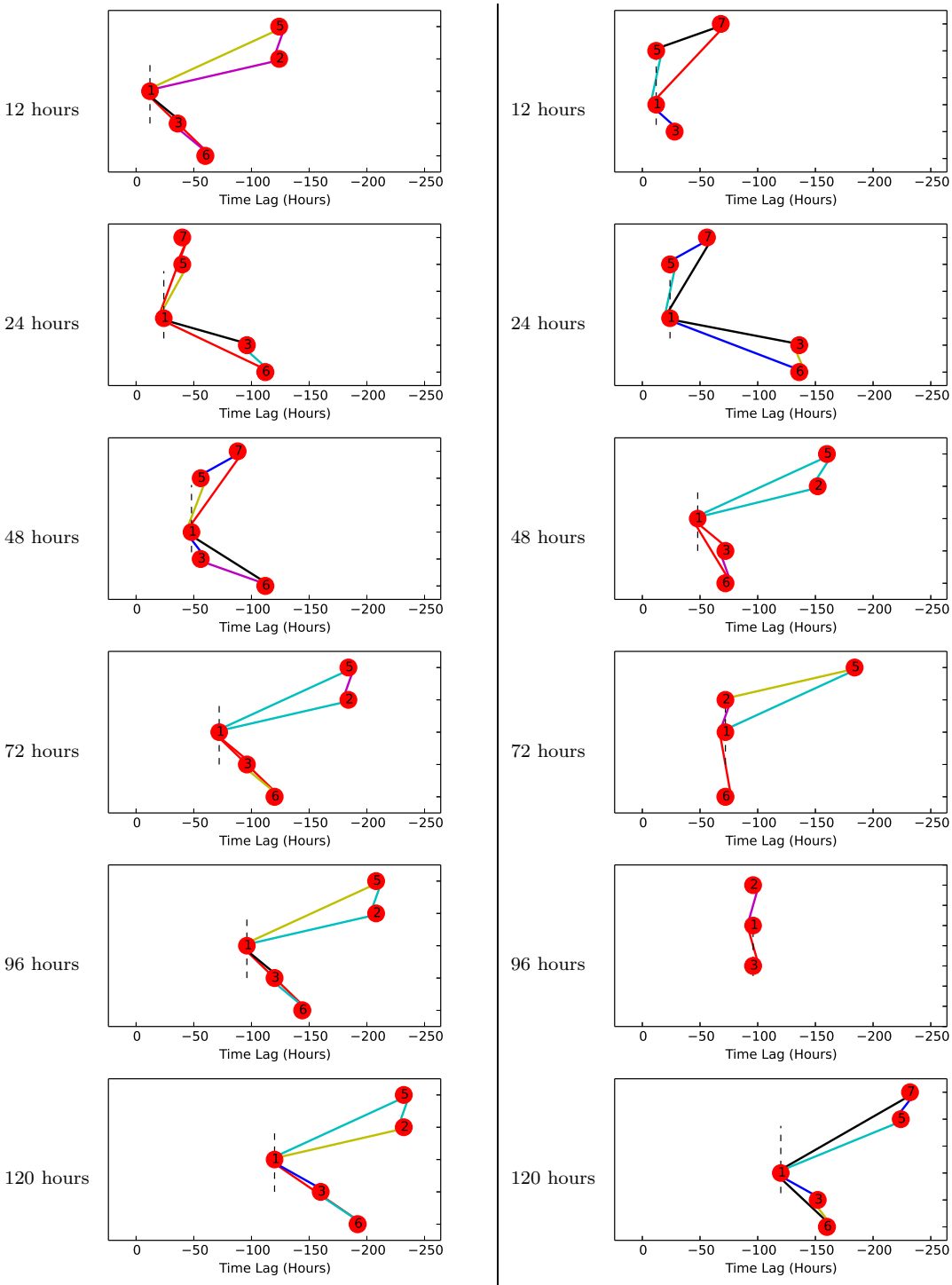
Table B.1: Verbose statistics from Table 6.2 for univariate methods.



(a) Exhaustive Feature Selection Search

(b) Greedy Feature Selection Search

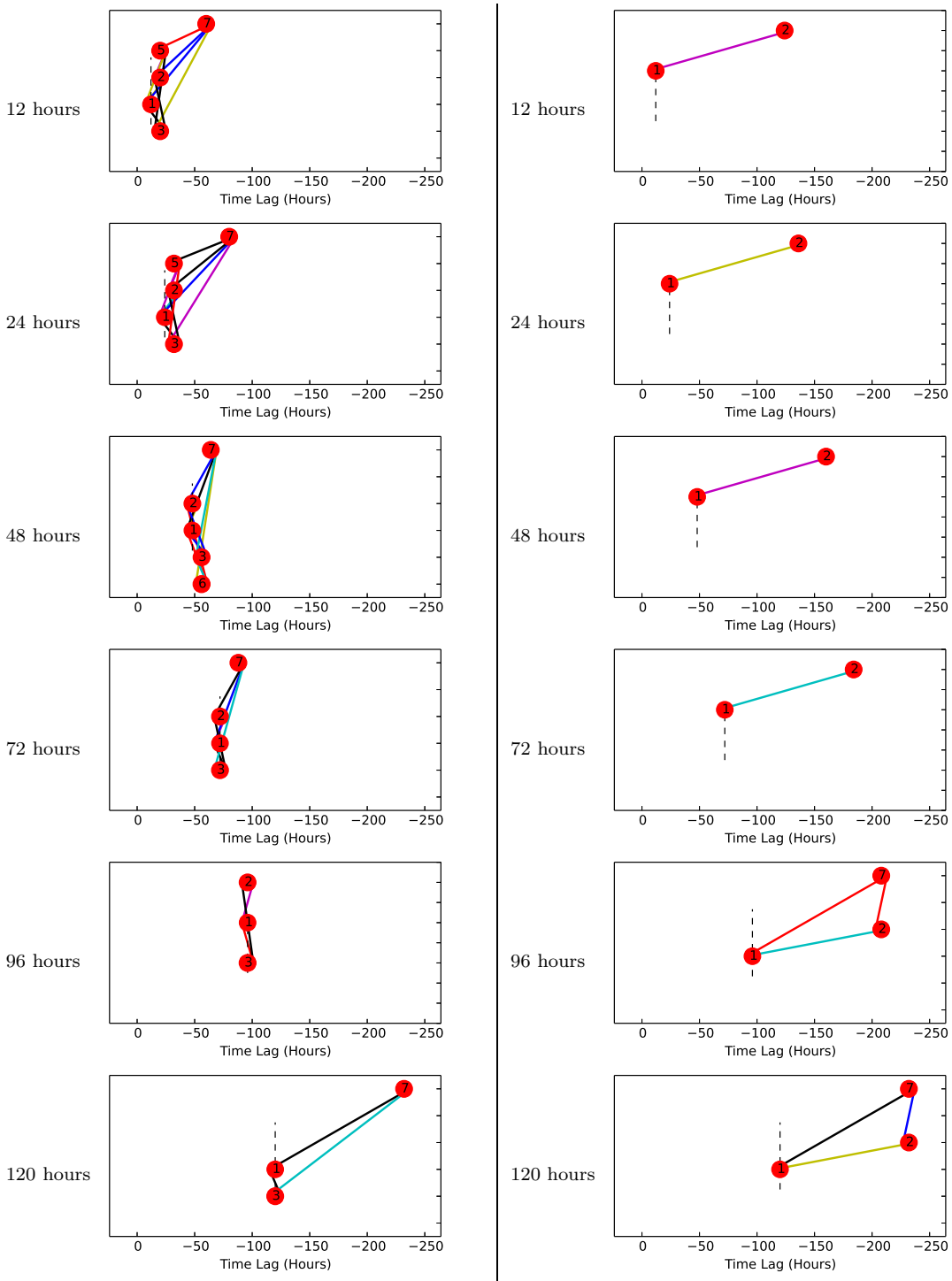
Figure B.1: Visualizations: Physical network derived constraints



(a) Random Feature Selection Search

(b) Guided Feature Selection Search

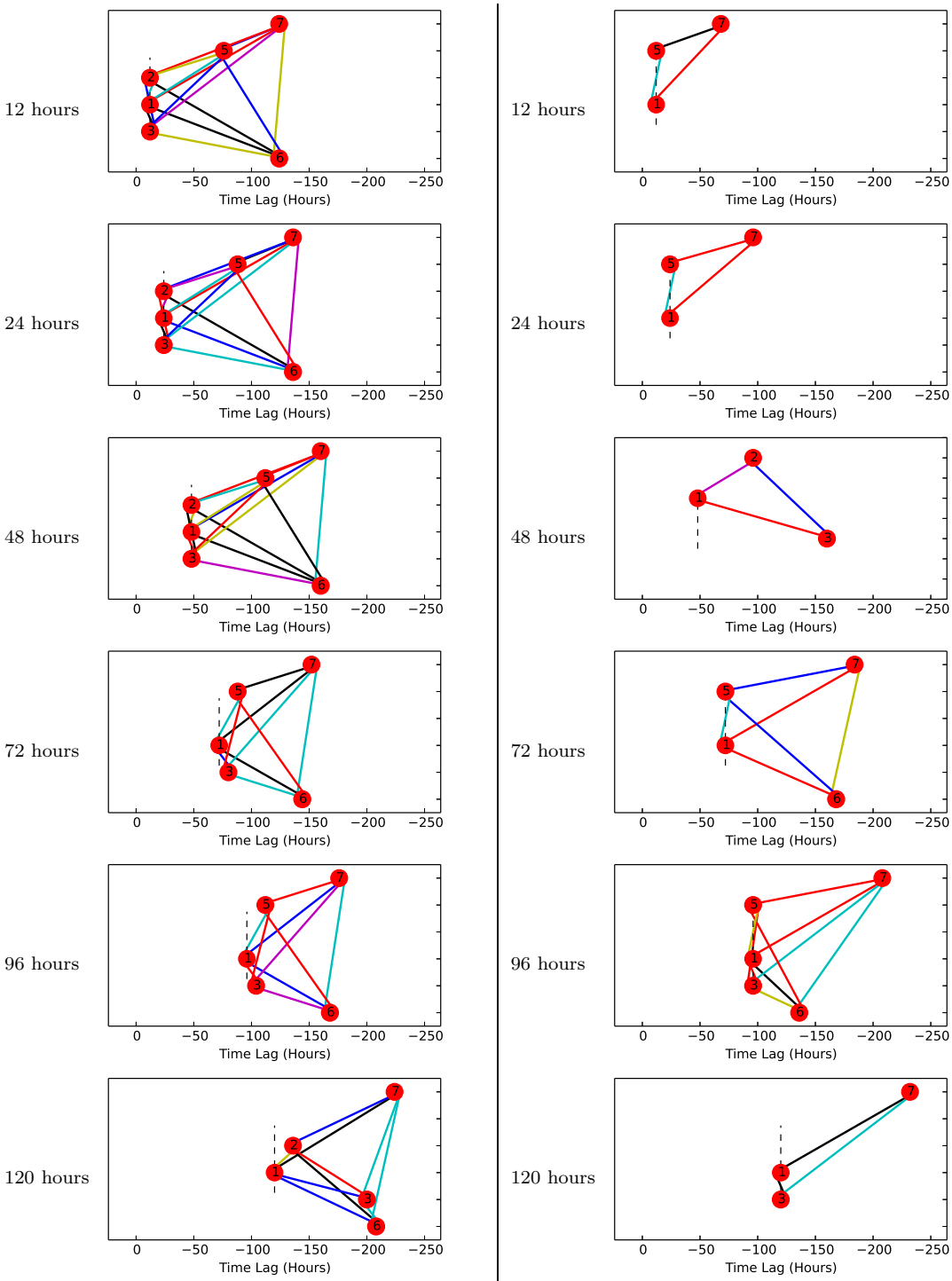
Figure B.2: Visualizations: Physical network derived constraints (continued)



(a) Exhaustive Feature Selection Search

(b) Greedy Feature Selection Search

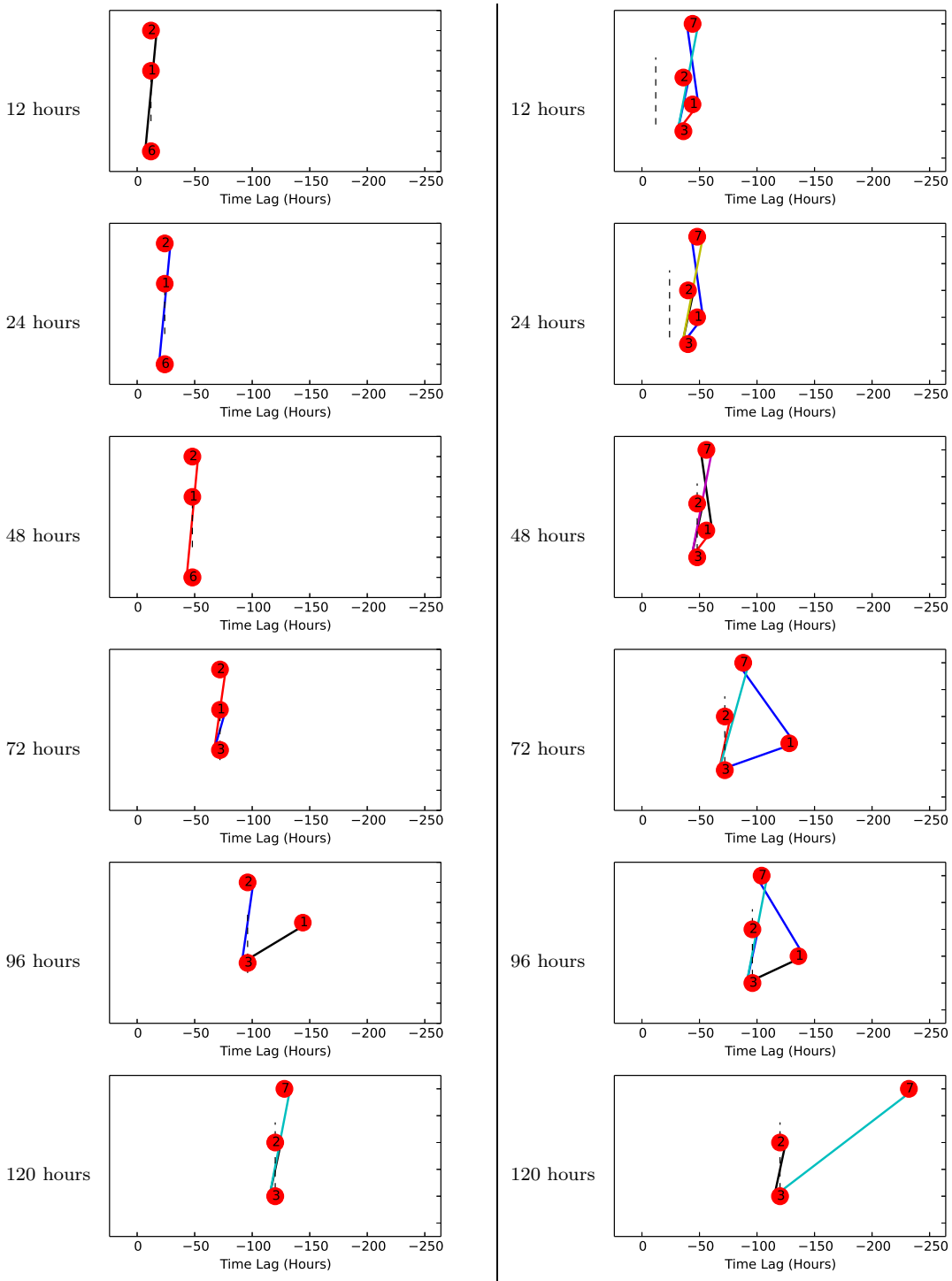
Figure B.3: Visualizations: Bad Network I derived constraints



(a) Random Feature Selection Search

(b) Guided Feature Selection Search

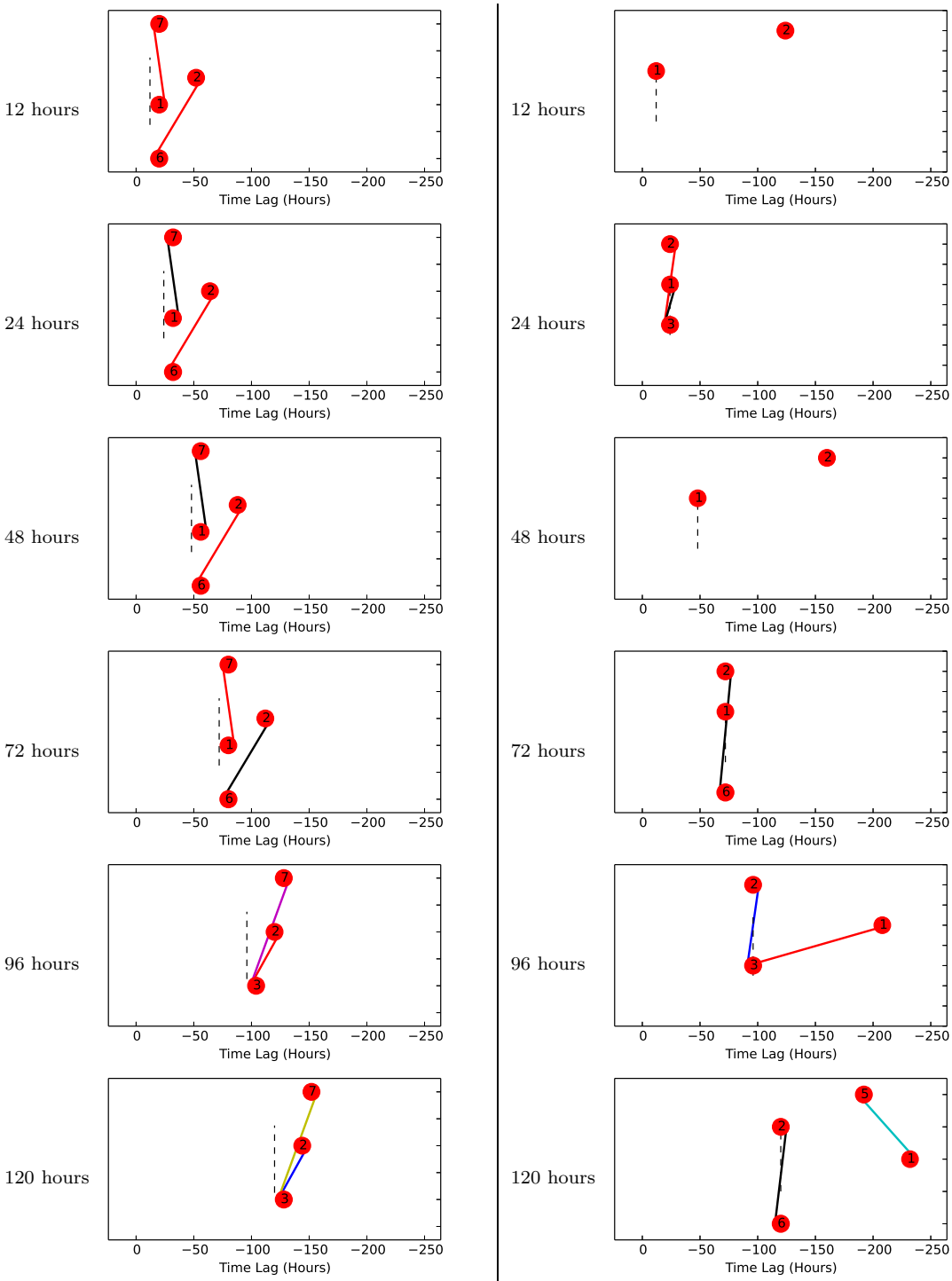
Figure B.4: Visualizations: Bad Network I derived constraints (continued)



(a) Exhaustive Feature Selection Search

(b) Greedy Feature Selection Search

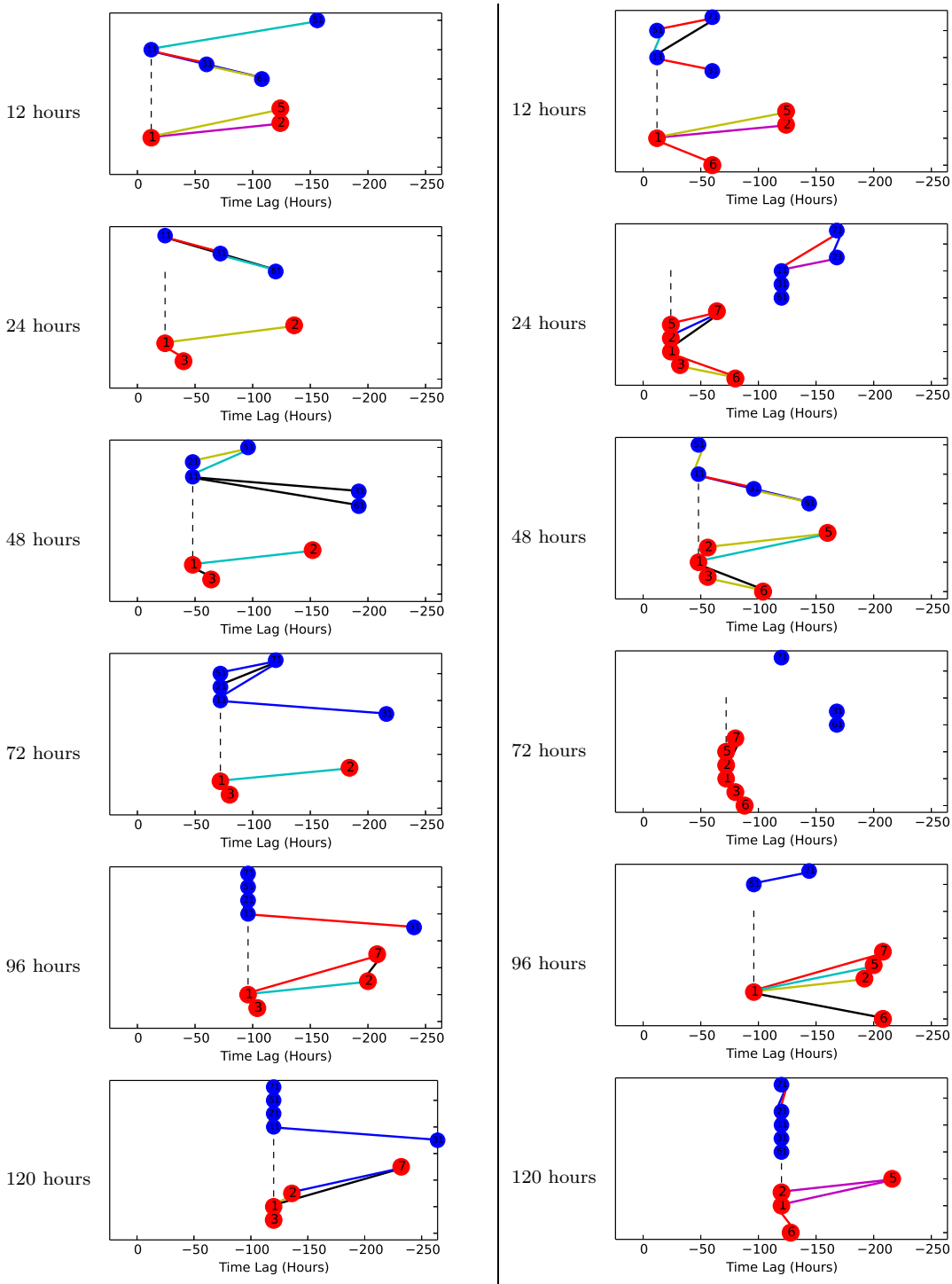
Figure B.5: Visualizations: Bad Network II derived constraints



(a) Random Feature Selection Search

(b) Guided Feature Selection Search

Figure B.6: Visualizations: Bad network II derived constraints (continued)



(a) Greedy Feature Selection Search

(b) Guided Feature Selection Search

Figure B.7: Visualizations: Physical network derived constraints with precipitation data

score_{va} score_{te} runtime (secs) # of Vars		**Target Offset**					
Method	**Algorithm**	12	24	48	72	96	120
fsALL	PLS_BFSB	47.5 63.4 3222 90.0	85.7 105.2 6201 90.0	88.2 104.7 4637 90.0	134.8 138.8 3598 90.0	223.2 235.2 2239 90.0	215.4 220.9 3195 90.0
fsALL	PLS_BFSF	201.1 260.6 70 90.0	200.8 260.3 69 90.0	200.4 259.9 70 90.0	200.0 259.7 70 90.0	199.7 259.5 68 90.0	199.4 259.5 69 90.0
fsALL	PLS_CFSB	29.6 34.6 30 90.0	34.6 38.5 30 90.0	59.3 62.0 37 90.0	71.3 80.9 39 90.0	97.7 108.1 32 90.0	116.0 135.2 32 90.0
fsALL	PLS_CFSF	40.7 49.6 25 90.0	36.2 40.5 23 90.0	73.1 81.7 24 90.0	83.6 93.1 25 90.0	99.7 112.8 24 90.0	118.1 136.1 25 90.0
fsALL	REPTree	68.0 78.1 19 90.0	73.7 83.0 36 90.0	147.8 110.5 37 90.0	154.4 109.9 36 90.0	126.7 142.3 36 90.0	134.6 138.6 36 90.0
fsALL	REPTree_BFSF	86.0 83.4 2616 90.0	88.2 107.9 1580 90.0	93.6 94.6 852 90.0	163.2 121.6 2533 90.0	155.6 107.6 2706 90.0	157.6 139.5 1749 90.0
fsALL	REPTree_CFSF	51.7 62.2 43 90.0	83.0 89.0 44 90.0	101.4 114.0 23 90.0	114.6 96.5 50 90.0	111.9 127.8 45 90.0	140.6 134.3 46 90.0
fsALL	PLS	16.9 19.5 14 90.0	26.3 29.9 13 90.0	52.6 60.5 13 90.0	89.6 104.2 14 90.0	124.4 147.4 15 90.0	154.2 183.8 16 90.0
fsALL	Ridge	22.8 26.2 14 90.0	50.6 55.8 24 90.0	119.8 120.8 26 90.0	213.8 215.9 26 90.0	310.8 319.7 26 90.0	372.3 391.3 25 90.0
fsALL	nuSVR	14.6 17.6 1491 90.0	31.3 39.3 1995 90.0	62.7 72.5 1239 90.0	116.8 113.1 897 90.0	183.1 157.1 881 90.0	249.3 206.4 797 90.0
mostrecent	REPTree	73.9 87.2 17 6.0	77.4 89.3 18 6.0	99.9 81.9 17 6.0	138.7 97.2 17 6.0	182.4 113.2 18 6.0	161.6 126.6 19 6.0
mostrecent	PLS	16.9 19.8 15 6.0	35.1 41.8 18 6.0	61.2 72.1 15 6.0	72.9 83.4 14 6.0	82.3 92.5 17 6.0	92.8 106.0 14 6.0
mostrecent	Ridge	16.9 19.8 16 6.0	35.1 41.8 14 6.0	61.2 72.1 15 6.0	72.9 83.4 15 6.0	82.3 92.5 15 6.0	92.8 106.0 16 6.0
mostrecent	nuSVR	12.4 14.1 63 6.0	26.2 31.7 55 6.0	50.8 62.1 52 6.0	66.7 81.6 58 6.0	79.4 95.2 67 6.0	92.9 110.8 62 6.0
mostrecent	skRF	67.9 78.1 18 6.0	63.6 74.6 19 6.0	92.6 80.9 18 6.0	126.2 96.0 18 6.0	147.2 123.3 18 6.0	152.3 124.3 19 6.0

Table B.2: Verbose statistics from Tables 6.4, 6.6, and 6.8 for multivariate methods (both without and with temporal data).

score_{va} score_{te} runtime (secs) # of Vars		**Target Offset**					
Method	**Algorithm**	12	24	48	72	96	120
NOAApredictions	N/A	10.0 10.8 18 1.0	11.7 13.8 17 1.0	17.5 20.9 17 1.0	24.4 28.9 17 1.0	33.7 40.7 17 1.0	45.9 57.7 17 1.0
fsExhaustive	PLS	10.7 10.2 6659 4.0	19.3 19.0 6499 4.0	36.2 38.2 6976 4.0	53.2 58.5 7046 4.0	69.4 78.1 7870 4.0	84.5 96.3 7637 4.0
fsGreedy	PLS	12.0 12.8 24 3.0	22.3 24.7 19 3.0	44.1 47.9 32 3.0	59.9 66.7 33 3.0	67.8 75.1 34 3.0	81.6 92.7 34 4.0
fsGuided	PLS	13.4 11.6 196 4.0	25.4 22.4 174 5.0	41.9 47.8 306 5.0	53.2 58.5 323 4.0	67.8 75.1 315 3.0	88.6 105.3 338 5.0
fsRandom	PLS	11.9 13.0 128 5.0	26.3 26.0 201 5.0	54.9 50.6 216 5.0	62.5 73.0 247 5.0	76.6 89.9 242 5.0	91.6 107.8 237 5.0
precip+fsGreedy	PLS	10.9 11.0 41 35.0	20.6 20.6 60 27.0	37.9 39.8 69 43.0	56.7 55.9 66 43.0	67.7 68.9 71 44.0	76.3 81.7 77 44.0
precip+fsGuided	PLS	10.6 10.9 408 36.0	21.6 20.5 734 46.0	36.7 37.3 742 37.0	61.0 66.6 682 30.0	83.5 81.5 628 21.0	77.1 86.6 718 44.0
precip+fsRandom	PLS	12.2 12.0 544 38.0	23.1 23.0 714 38.0	49.0 46.4 707 38.0	64.7 73.0 692 53.0	77.0 83.7 667 35.0	89.2 97.1 672 35.0

Table B.3: Verbose statistics from Table 6.11: DGFS feature selection methods on ground truth constraint network (A).

score$_{va}$ score$_{te}$ runtime (secs) # of Vars		**Target Offset**					
Network	**Method**	12	24	48	72	96	120
	NOAApredictions	10.0 10.8 18 1.0	11.7 13.8 17 1.0	17.5 20.9 17 1.0	24.4 28.9 18 1.0	33.7 40.7 17 1.0	45.9 57.7 17 1.0
A	fsExhaustive	10.7 10.2 6659 4.0	19.3 19.0 6499 4.0	36.2 38.2 6976 4.0	53.2 58.5 7046 4.0	69.4 78.1 7870 4.0	84.5 96.3 7637 4.0
A	fsGreedy	12.0 12.8 24 3.0	22.3 24.7 19 3.0	44.1 47.9 32 3.0	59.9 66.7 33 3.0	67.8 75.1 34 3.0	81.6 92.7 34 4.0
A	fsGuided	13.4 11.6 196 4.0	25.4 22.4 174 5.0	41.9 47.8 306 5.0	53.2 58.5 323 4.0	67.8 75.1 315 3.0	88.6 105.3 338 5.0
A	fsRandom	11.9 13.0 128 5.0	26.3 26.0 201 5.0	54.9 50.6 216 5.0	62.5 73.0 247 5.0	76.6 89.9 242 5.0	91.6 107.8 237 5.0
B	fsExhaustive	66.4 55.9 1585 3.0	65.3 56.1 1921 3.0	67.5 59.1 2647 3.0	67.8 71.1 1900 3.0	79.6 88.0 2727 2.0	88.2 100.2 3327 3.0
B	fsGreedy	11.9 12.9 16 2.0	22.3 24.9 23 2.0	42.5 49.0 17 2.0	61.1 72.2 17 2.0	79.0 91.6 21 3.0	85.2 104.4 17 3.0
B	fsGuided	13.3 11.8 188 3.0	30.0 23.0 209 3.0	41.4 47.9 173 3.0	84.9 70.8 191 4.0	100.1 93.3 215 5.0	84.9 93.6 183 3.0
B	fsRandom	14.4 12.9 81 6.0	29.4 26.9 84 6.0	56.8 54.0 69 6.0	100.1 79.6 83 5.0	118.9 100.1 76 5.0	99.5 119.8 89 5.0
C	fsExhaustive	66.4 55.9 1585 3.0	65.3 56.1 1921 3.0	67.5 59.1 2647 3.0	67.8 71.1 1900 3.0	79.6 88.0 2727 2.0	88.2 100.2 3327 3.0
C	fsGreedy	48.3 49.6 18 4.0	50.9 52.6 26 4.0	55.5 57.5 16 4.0	58.6 61.7 18 4.0	68.9 72.8 27 4.0	79.5 87.2 18 3.0
C	fsGuided	11.9 12.9 155 2.0	24.9 26.3 187 3.0	42.5 49.0 151 2.0	58.9 65.4 146 3.0	68.3 72.0 215 3.0	84.5 95.2 149 4.0
C	fsRandom	27.2 28.7 30 4.0	39.2 42.8 44 4.0	61.3 68.9 35 4.0	74.1 86.4 33 4.0	89.5 93.9 47 3.0	98.4 104.2 34 3.0
	lastvalue	12.9 14.6 31 1.0	24.1 27.9 28 1.0	44.8 52.8 30 1.0	63.2 75.5 16 1.0	79.1 95.6 14 1.0	92.7 112.7 14 1.0

Table B.4: Verbose statistics from Table 6.15: DGFS feature selection methods over constraint networks containing mistakes (A, B, and C).

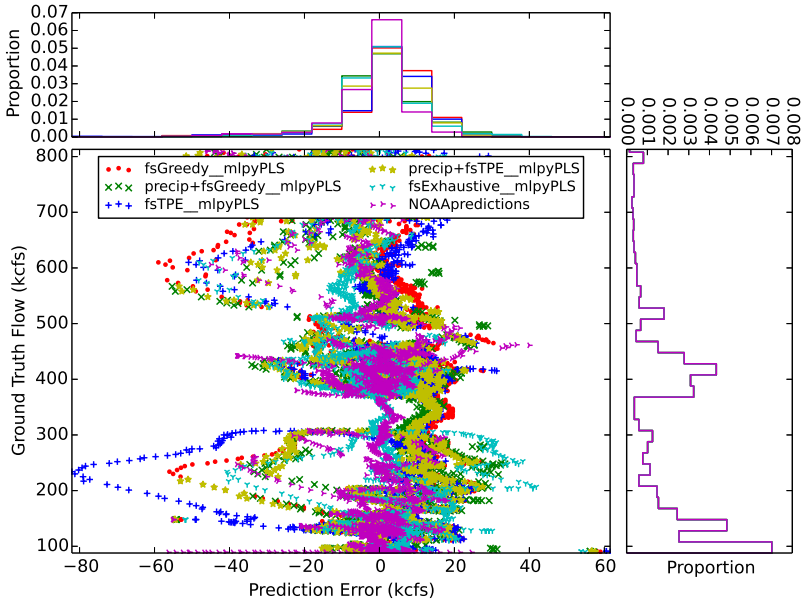


Figure B.8: EADM7 2d histogram comparing DGFS methods at 12 hour prediction target.

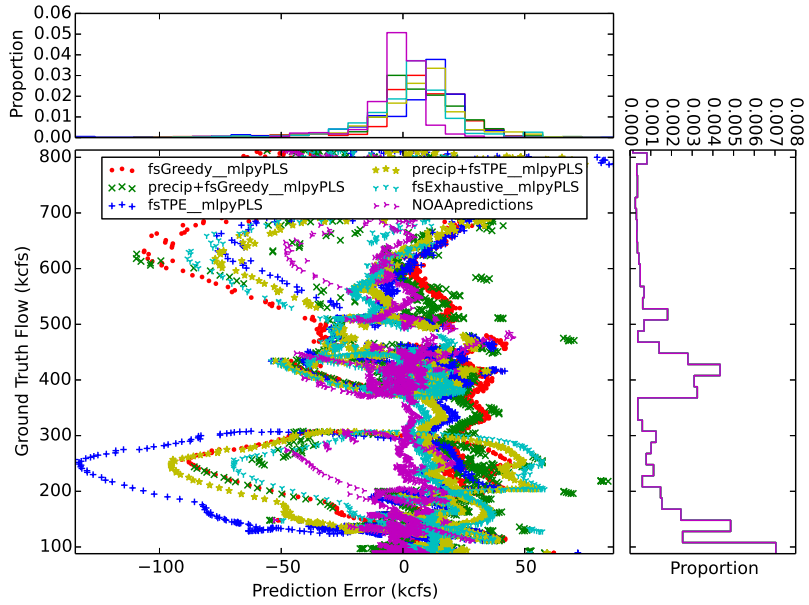


Figure B.9: EADM7 2d histogram comparing DGFS methods at 24 hour prediction target.

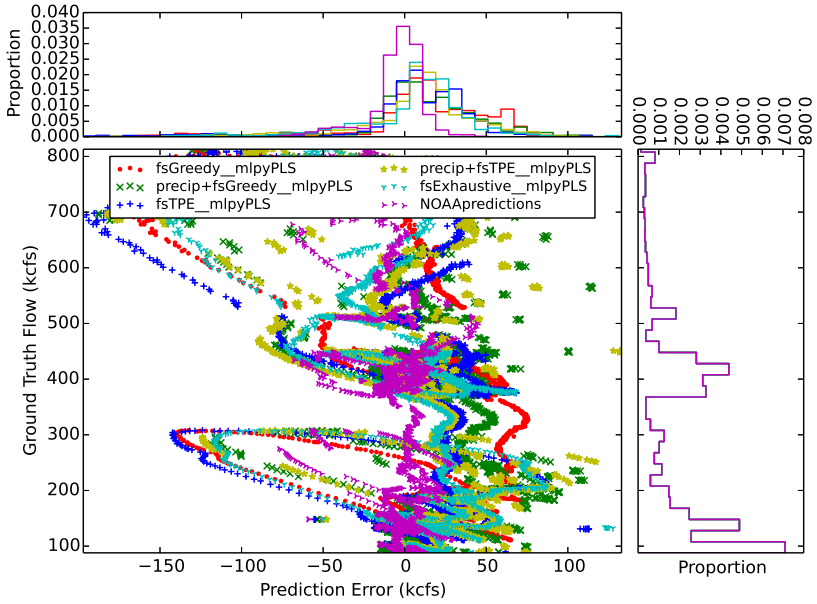


Figure B.10: EADM7 2d histogram comparing DGFS methods at 48 hour prediction target.

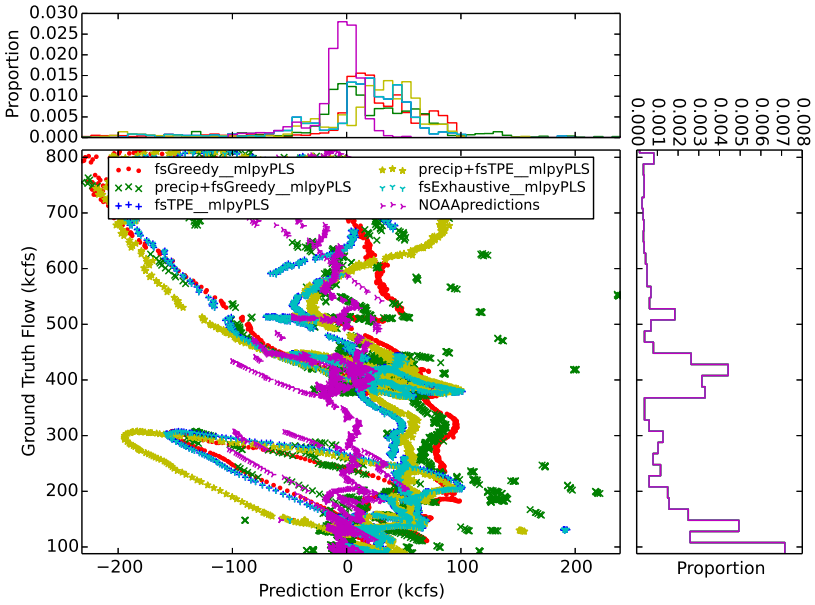


Figure B.11: EADM7 2d histogram comparing DGFS methods at 72 hour prediction target.

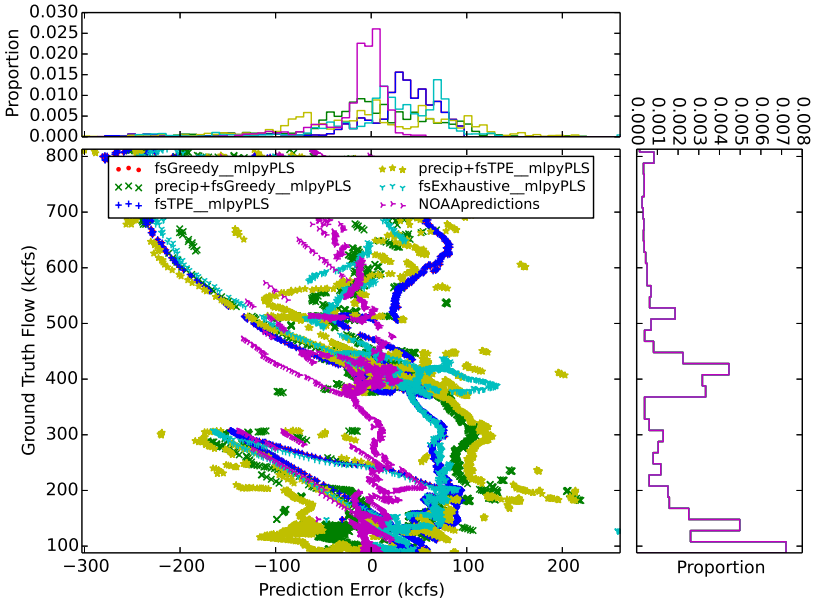


Figure B.12: EADM7 2d histogram comparing DGFS methods at 96 hour prediction target.

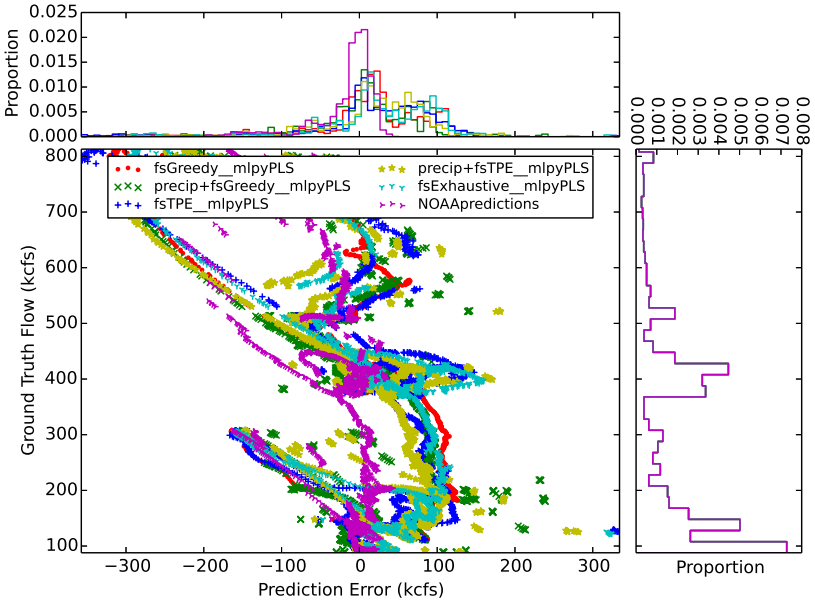


Figure B.13: EADM7 2d histogram comparing DGFS methods at 120 hour prediction target.