

INSTANCE-BASED REGRESSION
BY PARTITIONING FEATURE
PROJECTIONS

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

BY
ILHAN UYSAL
JANUARY, 2000

THESIS
QA
278.2
'497
2000

INSTANCE-BASED REGRESSION BY PARTITIONING FEATURE PROJECTIONS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

by

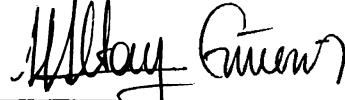
İlhan Uysal

January, 2000

QA
278.2
U37
2000

B 051127

I certify that I have read this thesis and that in my opinion it is fully adequate,
in scope and in quality, as a thesis for the degree of Master of Science.



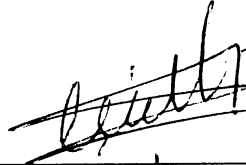
Assoc. Prof. Dr. Halil Altay Güvenir (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate,
in scope and in quality, as a thesis for the degree of Master of Science.



Assoc. Prof. Dr. Kemal Oflazer

I certify that I have read this thesis and that in my opinion it is fully adequate,
in scope and in quality, as a thesis for the degree of Master of Science.



Asst. Prof. Dr. İlyas Çiçekli

Approved for the Institute of Engineering and Science:



Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

INSTANCE-BASED REGRESSION BY PARTITIONING FEATURE PROJECTIONS

İlhan Uysal

M.S. in Computer Engineering

Supervisor: Assoc. Prof. Halil Altay Güvenir

January, 2000

A new instance-based learning method is presented for regression problems with high-dimensional data. As an instance-based approach, the conventional K-Nearest Neighbor (KNN) method has been applied to both classification and regression problems. Although KNN performs well for classification tasks, it does not perform similarly for regression problems. We have developed a new instance-based method, called Regression by Partitioning Feature Projections (RPFP), to fill the gap in the literature for a lazy method that achieves a higher accuracy for regression problems. We also present some additional properties and even better performance when compared to famous eager approaches of machine learning and statistics literature such as MARS, rule-based regression, and regression tree induction systems. The most important property of RPFP is that it performs much better than all other eager or lazy approaches on many domains that have missing values. If we consider databases today, where there are generally large number of attributes, such sparse domains are very frequent. RPFP handles such missing values in a very natural way, since it does not require all the attribute values to be present in the data set.

Keywords: Machine learning, instance-based learning, regression.

ÖZET

ÖZİNİTELİK İZDÜŞÜMLERİNİN PARÇALANMASI İLE ÖRNEKLERE DAYALI REGRESYON

İlhan Uysal

Bilgisayar Mühendisliği, Yüksek Lisans

Tez Yöneticisi: Doç. Dr. Halil Altay Güvenir

Ocak, 2000

Yüksek öznitelik sayılarına sahip verilerin regresyon çözümleri için örneklere dayalı yeni bir öğrenme metodu sunulmuştur. Örneklere dayalı bir yaklaşım olarak geleneksel K-Yakın Komşu (KNN) yöntemi hem sınıflandırma hem de regresyon problemleri için uygulanmıştır. KNN sınıflandırma işlemleri için iyi bir performans sergilerken, regresyon için benzer bir performansa sahip değildir. Biz literatürdeki bu boşluğu doldurmak üzere, tembel öğrenme yaparak yüksek başarı sağlayan örneklere dayalı yeni bir regresyon yöntemi olan, Öznitelik İzdüşümlerinin Parçalanması ile Regresyon (RPFİ) isimli yöntemi geliştirdik. RPFİ makina öğrenmesi ve istatistik literatüründe yer alan MARS, kurallara dayalı regresyon ve regresyon ağacı öğrenen sistemler gibi önemli çalışkan algoritmalarda dahi bulunmayan bazı özelliklere ve hatta daha iyi performansa sahiptir. RPFİ'nin bu özelliklerinden en önemli olanı verilerde eksik değerler olduğu durumlarda pek çok 'uygulama için diğer tüm çalışkan veya tembel yöntemlerden daha çok başarı sağlamasıdır. Günümüzde, çok sayıda alanları bulunan veri tabanlarını dikkate aldığımız zaman, böyle ortamlara sıklıkla rastlanır. RPFİ veri seti içindeki tüm öznitelik değerlerinin doldurulmuş olmasını gerektirmediği için eksik olan değerleri doğal bir şekilde çözümler.

Anahtar Sözcükler: Makina öğrenmesi, örneklere dayalı öğrenme, regresyon.

Babama ve anneme.

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. H. Altay Güvenir, from whom I have learned a lot, due to his supervision, suggestions, and support during this research.

I am also indebted to Dr. Kemal Ofłazer and Dr. İlyas Çiçekli for showing keen interest to the subject matter and accepting to read and review this thesis.

I would like to thank to Yeşim and Gözde, for their great patience and support that was very important, Halise for her support and data sets she provided, Nurhan for her morale support and Munise for many things.

I would also like to thank to Bilkent University Computer Engineering Department and Turkish Navy, since they enabled and supported this research.

This thesis was supported, in part, by TUBITAK (Scientific and Technical Research Council of Turkey) under Grant 198E015.

Contents

1	Introduction	1
1.1	Parametric versus Non-Parametric Regression	3
1.2	Eager versus Lazy Learning	4
1.3	Regression by Partitioning Feature Projections	5
1.4	Notation	6
1.5	Organization	6
2	Overview of Regression Techniques	7
2.1	Instance-Based Regression	7
2.2	Locally Weighted Regression	9
2.2.1	Nonlinear Local Models	10
2.2.2	Linear Local Models	10
2.2.3	Implementation	11
2.3	Regression by Rule Induction	12
2.4	Projection Pursuit Regression	16
2.4.1	Projection Pursuit Regression Algorithm	17

2.4.2	Smoothing Algorithm	19
2.5	Regression by Tree Induction	20
2.5.1	CART	21
2.5.2	RETIS	26
2.5.3	M5	28
2.6	Multivariate Adaptive Regression Splines	29
2.6.1	Piecewise Parametric Fitting and Splines	30
2.6.2	MARS Algorithm	30
2.7	Discussion	33
3	Regression by Partitioning Feature Projections	36
3.1	RFPF Algorithm	39
3.1.1	Training	39
3.1.2	Approximation using Feature Projections	40
3.1.3	Local Weight	43
3.1.4	Partitioning Algorithm	45
3.1.5	Prediction	48
3.2	RFPF-N Algorithm	49
3.3	Properties of RFPF	51
3.3.1	Curse of Dimensionality	51
3.3.2	Bias-variance Trade-off	54
3.3.3	Model Complexity and Occam's Razor	55

3.3.4	Lazy Learning	56
3.3.5	Local Learning	57
3.3.6	Irrelevant Features and Dimensionality	58
3.3.7	Context-sensitive Learning	59
3.3.8	Missing Feature Values	60
3.3.9	Interactions	61
3.3.10	Feature Projection Based Learning	61
3.3.11	Different Feature Types	62
3.3.12	Noise	62
3.3.13	Normalization	63
3.4	Limitations of RFPF	63
3.4.1	Redundant Features	63
3.4.2	Interpretation	64
3.4.3	Rectangular Regions	64
3.5	Complexity Analysis	65
3.6	Comparisons of Regression Methods	65
4	Empirical Evaluations	67
4.1	Performance Measure	68
4.2	Algorithms Used in Comparisons	68
4.2.1	RFPF	69
4.2.2	KNN	69

4.2.3	RULE	69
4.2.4	DART	69
4.2.5	MARS	70
4.3	Real Data Sets	70
4.4	Accuracy	70
4.5	Robustness to Irrelevant Features	72
4.6	Robustness to Missing Values	75
4.7	Robustness to Noise	75
4.8	Interactions.	79
4.9	Computation Times	82
5	Conclusion and Future Work	84

List of Figures

2.1	The Proximity Algorithm	8
2.2	<i>2d-tree</i> data structure. The black dot is the query point, and the shaded dot is the nearest neighbor. Outside the black box does not need to be searched to find the nearest neighbor.	12
2.3	Swap-1 Algorithm	13
2.4	A solution induced from a hart-disease data	14
2.5	Composing Pseudo-Classes (P-Class)	15
2.6	Overview of Method for Learning Regression Rules	16
2.7	Projection Pursuit Regression Algorithm	18
2.8	Running Medians of Three	19
2.9	Variable Bandwidth Smoothing Algorithm	20
2.10	Example of Regression Tree	22
2.11	An example of the tree construction of process. Four regions are determined by predictors x_1 and x_2	23
2.12	Recursive Partitioning Algorithm	24
2.13	A binary tree representing a recursive partitioning regression model with the associated basis functions	25

2.14	An example region, with large variance, which is inappropriate for splitting	26
2.15	MARS Algorithm	33
2.16	An example for the regions of MARS algorithm	33
3.1	An example training set projected to two features: f_1 and f_2 . . .	40
3.2	Approximations for Feature Projections	42
3.3	Example data set and its partitioning.	46
3.4	Partitioning Algorithm	47
3.5	Example training set after partitioning.	49
3.6	Approximations for Feature Projections	49
3.7	Prediction Algorithm	50
3.8	Weighted Median Algorithm	51
4.1	Relative errors of algorithms with increasing irrelevant features.	74
4.2	Relative errors of algorithms with increasing missing values.	77
4.3	Relative errors of algorithms with increasing target noise.	80
4.4	Artificial data set. x_1 and x_2 are input features.	81

List of Tables

2.1	Example of swapping rule components.	14
2.2	Properties of Regression Algorithms (the names of programs developed with those methods are shown in parentheses).	35
3.1	Properties of Regression Algorithms. The (\checkmark) is used for cases if the corresponding algorithm handles a problem or it is advantageous in a property when compared to others.	66
4.1	Characteristics of the data sets used in the empirical evaluations. C: Continuous, N: Nominal.	71
4.2	Relative Errors of Algorithms. Best results are typed with bold font.	73
4.3	Relative errors of algorithms, where 30 irrelevant features are added to real data sets. If the result is not available due to singular variance/covariance matrix, it is shown with (*). Best results are typed with bold font.	76
4.4	Relative errors of algorithms, where 20% of values of real data sets are removed. If the result is not available due to singular variance/covariance matrix, it is shown with (*). Best results are typed with bold font.	78
4.5	Comparison of RFPF with its additive version RFPF-A. Best results are typed with bold font.	81

4.6 Time durations of algorithms for real data sets in milliseconds. . 83

List of Symbols and Abbreviations

B	: Basis Function
β	: Parameter set
CART	: Classification and Regression Trees
CFP	: Classification by Feature Partitioning
COFI	: Classification by Overlapping Feature Intervals
D	: Training Set
DART	: Regression tree induction algorithm
DNF	: Disjunctive Normal Form
f	: Approximated Function
g	: Approximated Function
H	: Step Function
I	: Fitting Function
i	: Instance
IBL	: Instance-based Learning
K	: Kernel Function
k	: Number of Neighbor Instances
KMEANS	: Partitioning clustering algorithm
KNN	: K Nearest Neighbor
KNNFP	: K Nearest Neighbor on Feature Projections
KNNR	: K Nearest Neighbor Regression
KDD	: Knowledge Discovery in Databases
L	: Loss Function
LOESS	: Locally weighted regression program
LOF	: Lock of Fit Function
log	: Logarithm in base 2
LW	: Local Weight
LW_f	: Local weight of feature f
LWR	: Locally Weighted Regression
MAD	: Mean Absolute Distance
MARS	: Multivariate Adaptive Regression Splines
M5	: Regression tree induction algorithm
n	: Number of training instances

p	: Number of parameters or features
PI	: Prediction Index
PPR	: Projection Pursuit Regression
q	: Query instance
x_q	: Query instance
R	: Region
R_k	: Rule set
RE	: Relative Error
RETIS	: Regression tree induction algorithm
RFP	: Regression by Feature Projections
RFPF	: Regression by Partitioning Feature Projections
RULE	: Rule-based regression algorithm
r	: Rule
r_i	: Residual
\mathbf{r}	: Residual vector
S	: Smooth function
t	: A test example
T	: Number of test instances
VFI	: Voting Feature Intervals
\mathbf{X}	: Instance matrix
\mathbf{x}	: Instance vector
\mathbf{x}_i	: Value vector of i^{th} instance
WKNN	: Weighted k Nearest Neighbor Algorithm
\mathbf{y}	: Target vector
\bar{y}	: Estimated target

Chapter 1

Introduction

Predicting values of numeric or continuous attributes is called *regression* in the statistical literature, and it has been an active research area in this field. Predicting real values is also an important topic for machine learning. Most of the problems that humans learn to solve in real life such as sporting abilities are continuous. Dynamic control is a research area in machine learning. For example, learning to catch a ball moving in a three dimensional space, is an example of this problem, studied in robotics. In such applications machine learning algorithms are used to control robot motions, where the response to be predicted by the algorithm is a numeric or real-valued distance measure and direction. As an example of such problem, Salzberg and Aha proposed an instance-based learning algorithm for robot control task in order to improve a robot's physical abilities [4].

In machine learning, much research has been performed on classification, where the predicted feature is nominal or discrete. Regression differs from classification, in that the output or predicted feature in regression problems is continuous. Even though, much research is concentrated on classification in machine learning, recently the focus of the machine learning community has moved strongly towards regression, since a large number of real-life problems can be modeled as regression problems. Various names are used for this problem in the literature, such as *functional prediction*, *real value prediction*, *function approximation* and *continuous class learning*. We prefer its historical

name, *regression*, henceforth, for simplicity.

In designing expert systems, induction techniques developed in machine learning and statistics have become important especially for cases where domain expert is not available or the knowledge of experts is tacit or implicit [1, 42]. These techniques are also important to discover knowledge in cases where domain experts or formal domain knowledge is available but difficult to elicit [39]. Probably, the most important advantage of induction techniques is that they enable us to extract knowledge automatically.

By the term “knowledge”, we mean two types of information. One is the information used for prediction of a new case, given example cases; the other is the information used for extracting new rules about the domain which have not yet been discovered, by interpreting induced models. The techniques reviewed and developed in this thesis can be employed in such systems, when the underlying problem is formalized as a prediction of a continuous target attribute.

The idea behind using induction techniques, investigated particularly in machine learning literature, is widely accepted by a newly emerged discipline, Knowledge Discovery in Databases (KDD), which incorporates researchers from various disciplines [17, 18, 60]. The main source of information in this field is large databases. Since databases can store large amounts of data belonging to many different domains, the use of automatic methods such as induction for knowledge discovery is viable, because it is usually difficult to find an expert for each different domain or relation in databases. Today, database management systems enable only deductive querying. Incorporating an inductive component into such databases to discover knowledge from different domains automatically is a long-term expectation from this new field [32]. This particularly requires the cooperation of knowledge engineers and database experts. Such expectations make regression an important tool for the stand-alone or domain-specific KDD systems today and Knowledge and Data Discovery Management Systems [17, 60] in the future.

1.1 Parametric versus Non-Parametric Regression

The most common approach in regression is to fit the data to a global parametric function. Classical *linear regression* in statistical analysis is an example of parametric learning. This model involves a dependent variable y and predictor (independent) variables (x 's), and assumes that the value of y changes at a constant rate as the value of any independent variable changes. Thus the functional relationship between y and x 's is a straight line.

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i \quad (1.1)$$

The subscript i denotes the observations or instances, the second subscript designates p independent variables. There are $p+1$ parameters, $\beta_j, j = 0, \dots, p$, to be estimated. In the parametric model, the structure of the function is given, and the procedure estimates the values of the parameters, β_j , according to a fitting criterion. This criterion is generally a minimization of an error function for all data points in a training set. Very often this is a *least squares* criterion, which minimizes the sum of the squares of the prediction errors of the estimated linear function for all instances. The error term, ε_i , denotes the error of estimation for each instance i , and it is assumed to be normally distributed.

Parametric methods have been very successful when the assumed structure of the function is sufficiently close to the function which generated the data to be modeled. However, the aim in machine learning is to find a general structure rich enough to model a large portion of all possible functions. This idea leads us to non-parametric regression methods, where no assumption is made about the structure of the function or about the distribution of the error.

1.2 Eager versus Lazy Learning

We categorize regression algorithms with two classes, *eager* and *lazy* approaches. The term *eager* is used for learning systems that construct rigorous models. By constructing models, two types of knowledge, prediction and concept descriptions that enable interpretation can be addressed. By using induced models of many eager methods, interpretation of the underlying data can be obtained. Decision trees and decision rules are such models, that are reviewed. On the other hand, *lazy* approaches [3] do not construct models and delay processing to the prediction phase. In fact the model is the data itself. Because of these properties, some disadvantages of the lazy approach immediately become apparent. The most important of all is that the lazy approaches are not suitable for the first type of knowledge, interpretation, since the data itself is not a compact description when compared other models such as trees or rules. So, the major task of these methods is prediction. A second limitation is that they generally have to store the whole data in the memory, it may be impossible if the data is too big.

However, *lazy* approaches are very popular in the literature, due to some of their important properties. One of them is that they make predictions according to the local position of query instances. They can form complex decision boundaries in the instance space even when relatively little information is available, since they do not generalize the data by constructing global models. Another one is that learning in lazy approaches is very simple and fast, since it only involves storing the instances. Finally, they do not have to construct a new model, when a new instance is added to the data.

Besides these common characteristics of lazy approaches, however, the most significant problem with them is the one posed by irrelevant features. Some feature selection and feature weighting algorithms have been developed in the literature for this purpose. A review of many such algorithms can be found in literature [61]. However, these algorithms have also a common characteristic that they ignore the fact that some features may be relevant only in context. That is, some features may be important or relevant only in some regions of the instance space. This characteristic is known as *context-sensitive* or *adaptive* in

the literature. Even most eager approaches have this property, most of lazy approaches are not adaptive. Such important properties of surveyed important regression techniques are also discussed in Chapter 2.

1.3 Regression by Partitioning Feature Projections

This thesis describes a new instance-based regression method based-on feature projections called Regression by Partitioning Feature Projections (RPFPP). Previous feature projection-based learning algorithms are developed for classification tasks. The RPFPP method works similar to those methods, by making predictions on the projections of data to all features separately. A complete survey of literature for feature projection-based learning is given in [13],

The RPFPP method described in thesis is adaptive. This property is also called as *context sensitive* in the literature. For different query locations in the instance space RPFPP forms a different model and a different region, and makes a different approximation. This is one of the major properties that makes RPFPP a flexible regression method. This brings in another advantage: *Robustness to irrelevant features*, as well as, eager algorithms that partition the instance space, such as, decision tree induction methods. The regions formed for the queries will be long on the dimensions of irrelevant features and short on relevant dimensions as the case in eager partitioning methods. Besides those properties, RPFPP is robust to the curse of dimensionality, in that it is suitable for high-dimensional data. This is due to the elimination of irrelevant features, and by making approximations on feature projections for each feature dimension separately. Making predictions on each feature separately enables another important property of RPFPP. It handles missing feature values naturally, without filling them with estimated values. The experimental results shows that, RPFPP achieves the highest accuracy when there are many missing values in the data set. These important properties of RPFPP and a detailed comparison of it with other famous approaches are described in detail after the description of RPFPP in Chapter 3.

From the point of view of these characteristics, we can define RPFPP as lazy, non-parametric, non-linear, and adaptive regression method based on feature projections in implementation.

1.4 Notation

In the rest of the thesis, training set D is represented by the instance matrix \mathbf{X} , where rows represent instances and columns represent predictors, and a response vector \mathbf{y} represents the continuous or numeric response to be predicted for all instances. Estimated values of y are shown with a column vector $\bar{\mathbf{y}}$, where \bar{y}_i is a scalar of the vector. Coefficients in Equation 1.1 are represented by a column vector $\boldsymbol{\beta}$. Any instance or any row in the instance matrix is represented by \mathbf{x}_i , where $i = 1, \dots, n$ and n is the number of instances in the training set. Any column of \mathbf{X} is represented by \mathbf{x}_j , where $j = 1, \dots, p$, and p is number of predictor features. x_{ij}, y_i and β_j represent scalars of \mathbf{X}, \mathbf{y} and $\boldsymbol{\beta}$, respectively. For the operations where $\boldsymbol{\beta}$ is included, a column consisting only of constant 1 values is inserted into the instance matrix as the first row so as to enforce the first term in Equation 1.1 ($j = 0, \dots, p$). The notations x_j and y are used as variables to represent predictor features and response feature respectively. To denote instance vectors (\mathbf{x}_i) with a variable, \mathbf{x} is used. To represent residuals, a column vector \mathbf{r} is used, where $r_i, i = 1, \dots, n$, is a scalar. To denote a query instance, a row vector \mathbf{q} or \mathbf{x}_q is used.

1.5 Organization

In next chapter, we make an overview of existing important regression techniques in the literature. In Chapter 3 we describe RPFPP and a robust version of it to noise RPFPP-N. The detailed description of characteristic properties of RPFPP and theoretical comparison of it with the existing important approaches in the literature is also given in this chapter. Empirical evaluations of RPFPP are shown in Chapter 4, and we conclude the thesis with Chapter 5.

Chapter 2

Overview of Regression Techniques

In this chapter, we review important regression techniques developed in machine learning and statistics. We first review lazy approaches for regression, instance-based regression, and locally weighted regression, in the first two sections and then we review eager approaches rule-based regression, projection pursuit regression, tree-based regression and multivariate adaptive regression splines, respectively in Section 2.3 through Section 2.6. We present a comparison of these techniques in Section 2.7 for their important properties.

2.1 Instance-Based Regression

Instance-based learning (IBL) algorithms are very popular since they are computationally simple during the training phase [2, 11]. In most applications, training is done simply by storing the instances in the memory. This section describes the application of this technique to regression problems [36].

In instance-based regression, each instance is usually represented as a set of attribute value pairs, where the values are either nominal or continuous, and the value to be predicted is continuous. Given query instance, the task is to predict the target value as a function of other similar instances whose target values are

known. The *nearest neighbor* is the most popular instance-based algorithm. The target values of the most similar neighbors are used in this task. Here the similarity is the complement of the Euclidean distance between instances. Formally, if we let real numbers, R be a numeric (continuous) domain, and \mathbf{X} be an instance space with p attributes, we can then describe the approximation function, F , for predicting numeric values as follows:

$$F(x_1, \dots, x_p) = \bar{y}_i \quad \text{where } \bar{y}_i \in R. \quad (2.1)$$

Training:

- [1] $\forall \mathbf{x}_i \in \text{Training Set}$
- [2] $\text{normalize}(\mathbf{x}_i)$

Testing:

- [1] $\forall \mathbf{x}_t \in \text{Test Set}$
- [2] $\text{normalize}(\mathbf{x}_t)$
- [3] $\forall \mathbf{x}_i \{\mathbf{x}_i \neq \mathbf{x}_t\}$: Calculate $\text{Similarity}(\mathbf{x}_t, \mathbf{x}_i)$
- [4] Let Similar s be set of N most similar instances to \mathbf{x}_t in Training Set
- [5] Let $\text{Sum} = \sum_{\mathbf{x}_i \in \text{Similar}s} \text{Similarity}(\mathbf{x}_t, \mathbf{x}_i)$
- [6] Then $\bar{y}_t = \sum_{\mathbf{x}_i \in \text{Similar}s} \frac{\text{Similarity}(\mathbf{x}_t, \mathbf{x}_i)}{\text{Sum}} F(\mathbf{x}_i)$

Figure 2.1. The Proximity Algorithm

There is a variety of instance-based algorithms in the literature. Here, the simplest one, the *proximity* algorithm is described in Figure 2.1. The proximity algorithm simply saves all training instances in the training set. The normalization algorithm maps each attribute value into the continuous range $(0 - 1)$. The estimate \bar{y}_t for test instance \mathbf{x}_t is defined in terms of a weighted similarity function of \mathbf{x}_t 's nearest neighbors in the training set. The similarity of two normalized instances is defined by Equation 2.2.

$$\text{Similarity}(\mathbf{x}_t, \mathbf{x}_i) = \sum_{j=1}^p \text{Sim}(x_{tj}, x_{ij}) \quad (2.2)$$

where $\text{Sim}(x, y) = 1.0 - |x - y|$ where $0 \leq x, y \leq 1$, and j is the feature dimension.

The assumption in this approach is that the function is locally linear. For sufficiently large sample sizes this technique yields a good approximation for continuous functions. Another important property of instance-based regression is its incremental learning behavior. By default, the instance-based regression assumes that all the features are equivalently relevant. However, the prediction accuracy of this technique can be improved by attaching weights to the attributes. To reduce the storage requirements for large training sets, averaging techniques for the instances can be employed [2]. The most important drawback of instance-based algorithms is that they do not yield abstractions or models that enable the interpretation of the training sets [40].

2.2 Locally Weighted Regression

Locally weighted regression (LWR) is similar to the nearest neighbor approach described in the previous section, especially for three main properties. First, the training phases of both algorithms include just storing the training data, and the main work is done during prediction. Such methods are also known as *lazy learning* methods. Secondly, they predict query instances with strong influence of the nearby or similar training instances. Thirdly, they represent instances as real-valued points in p -dimensional Euclidean space. The main difference between IBL and LWR is that, while the former predicts instances by averaging the nearby instances, the latter makes predictions by forming an averaging model at the location of query instance. This local model is generally a linear or nonlinear parametric function. After a prediction for query instance is done, this model is deleted, and for every new query a new local model is formed according to the location of the query instance. In such local models, nearby instances of the query have large weights on the model, whereas distant instances have fewer or no weights. For a detailed overview of the locally weighted methods see [7], from where the following subsections are summarized.

2.2.1 Nonlinear Local Models

Nonlinear local models can be constructed by modifying global parametric models. A general global model can be trained to minimize the following training criterion:

$$C = \sum_i L(f(\mathbf{x}_i, \boldsymbol{\beta}), y_i) \quad (2.3)$$

where y_i is the response value corresponding to the input vectors \mathbf{x}_i , and $\boldsymbol{\beta}$ is the parameter vector for the nonlinear model $\bar{y}_i = f(x_i, \boldsymbol{\beta})$ and L is the general loss function in predicting y_i . If this model is a neural net, then the $\boldsymbol{\beta}$ will be a vector of the synaptic weights. If we use the least squares for the loss function L , the training criterion will be

$$C = \sum_i (f(\mathbf{x}_i, \boldsymbol{\beta}) - y_i)^2 \quad (2.4)$$

In order to ensure points nearby to the query have more influence in the regression, a weighting factor can be added to the criterion.

$$C(q) = \sum_i [L(f(\mathbf{x}_i, \boldsymbol{\beta}), y_i) K(d(\mathbf{x}_i, \mathbf{q}))] \quad (2.5)$$

where K is the weighting or *kernel* function and $d(\mathbf{x}_i, \mathbf{q})$ is the distance between the data point \mathbf{x}_i and the query \mathbf{q} . Using this training criterion, f becomes a local model, and can have a different set of parameters for each query point.

2.2.2 Linear Local Models

The well-known linear global model for regression is simple regression (1.1), where least squares approximation is used as the training criterion. Such linear models can be expressed as

$$\mathbf{x}_i \boldsymbol{\beta} = y_i \quad (2.6)$$

where β is the parameter vector. Whole training data can be defined with the following matrix equation.

$$\mathbf{X}\beta = \mathbf{y} \quad (2.7)$$

where \mathbf{X} is the training matrix whose i th row is \mathbf{x}_i and \mathbf{y} is a vector whose i th element is y_i . Estimating the parameters β using the least squares criterion minimizes the following criterion:

$$C = \sum_i (\mathbf{x}_i\beta - y_i)^2 \quad (2.8)$$

We can use this global linear parametric model, where all the training instances have equal weight; for locally weighted regression, by giving nearby instances to the query point higher weights. This can be done using the following weighted training criterion:

$$C = \sum_i [(f(\mathbf{x}_i, \beta) - y_i)^2 K(d(\mathbf{x}_i, \mathbf{q}))]. \quad (2.9)$$

Various distance (d) and weighting (K) functions for local models are described in [7]. Different linear and nonlinear locally weighted regression models can be estimated using those functions.

2.2.3 Implementation

In LWR, as stated above, the computational cost of training is of a minimum since training includes only storing new data points into the memory. However the lookup procedure for prediction is more expensive than other instance-based learning methods, since a new model is constructed for each query. Here, the usage of a *kd-tree* data structure to speedup this process is described briefly [7].

The difficulty in the table lookup procedure is to find the nearest neighbors, if only nearby instances are included in LWR. If there are n instances in the

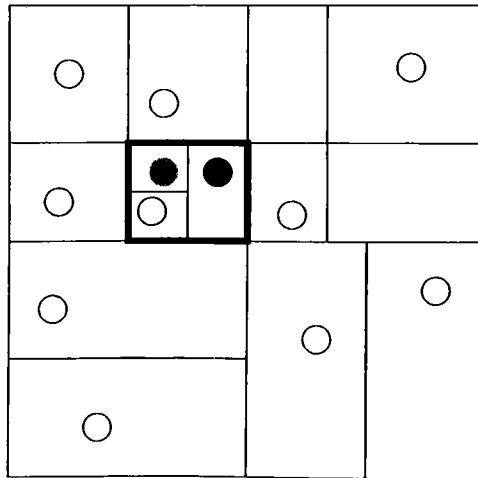


Figure 2.2. *2d-tree* data structure. The black dot is the query point, and the shaded dot is the nearest neighbor. Outside the black box does not need to be searched to find the nearest neighbor.

database, for a naive implementation we need n distance computations. For an efficient implementation, a *kd-tree* can be employed.

A *kd-tree* is a binary data structure that recursively splits a k -dimensional space into smaller subregions, and those subregions are the branches or leaves of the tree data structure. The search for the nearest neighbors starts from the nearby branches in the tree. For a given distance threshold there is no need to search further branches by implementing this data structure. Figure 2.2 illustrates a two-dimensional region.

2.3 Regression by Rule Induction

Inducing rules from a given training set is a well-studied topic in machine learning. Weiss and Indurkha employed rule induction for a regression problem and reported significant results [58, 59]. In this section, we will first review the rule-based classification algorithm [57], Swap-1, that learns decision rules in Disjunctive Normal Form (DNF), and later on describe its adaptation for regression.

```

[1] Input:  $D$ , a set of training cases
[2] Initialize  $R_1 \leftarrow$  empty set,  $k \leftarrow 1$ , and  $C_1 \leftarrow D$ 

[3] repeat
[4]   create a rule  $B$  with a randomly chosen attribute as its left-hand side
[5]   while ( $B$  is not 100-percent predictive) do
[6]     make single best swap for any component of  $B$ , including
       deletion of the component, using cases in  $C_k$ 
[7]     If no swap is found, add the single best component to  $B$ 
[8]   endwhile
[9]    $P_k \leftarrow$  rule  $B$  that is now 100-percent predictive
[10]   $E_k \leftarrow$  cases in  $C$  that satisfy the single-best-rule  $P_k$ 
[11]   $R_{k+1} \leftarrow R_k \cup \{P_k\}$ 
[12]   $C_{k+1} \leftarrow C_k - \{E_k\}$ 
[13]   $k \leftarrow k + 1$ 
[14] until ( $C_k$  is empty)
[15] find rule  $r$  in  $R_k$  that can be deleted without affecting performance
       on cases in training set  $D$ 
[16] while ( $r$  can be found)
[17]    $R_{k+1} \leftarrow R_k - \{r\}$ 
[18]    $k \leftarrow k + 1$ 
[19] endwhile
[20] output  $R_k$  and halt.

```

Figure 2.3. Swap-1 Algorithm

The main advantage of inducing rules in DNF is their explanatory capability. It is comparable to decision trees since they can also be converted into DNF models. The most important difference between them is that the rules are not mutually exclusive, as in decision trees. In decision trees, for each instance, there is exactly one rule encoding, a path from a root to a leaf, that is satisfied. Because of this restriction, decision tree models may not produce compact models. However, because of this property of rule-based models, the problem emerges that, for a single instance, two or more classes may be satisfied. The solution found for this problem is to assign priorities or ordering to the rules according to their extraction order. The first rule, according to this ordering that satisfies the query instance, determines the class of a query. The Swap-1 rule induction algorithm [57] and its sample output are shown in Figure 2.3 and Figure 2.4, respectively.

$$\begin{array}{ll}
 CA > 0.5 \text{ And } CP > 3.5 & \leftarrow \text{Class} = 2 \\
 THAL > 6.5 & \leftarrow \text{Class} = 2 \\
 [True] & \leftarrow \text{Class} = 1
 \end{array}$$

Figure 2.4. A solution induced from a hart-disease data

While constructing a rule, the Swap-1 algorithm searches all the conjunctive components it has already formed, and swaps them with all possible components it will build. This search also includes the deletion of some components from the rule. If no improvement is established from these swaps and deletions, then the best component is added to the rule. To find the best component to be added, the predictive value of a component, as the percentage of correct decisions, is evaluated. If the predictive values of them are equal, maximum instance coverage is used as the second criterion. These swappings and additions end when the rule reaches 100% prediction accuracy.

STEP	PREDICTIVE VALUE (%)	RULE
1	31	p3
2	36	p6
3	48	p6 & p1
4	49	p4 & p1
5	69	p4 & p1 & p2
6	80	p4 & p1 & p2 & p5
7	100	p3 & p1 & p2 & p5

Table 2.1. Example of swapping rule components.

Table 2.1 illustrates a sample rule induction. After forming a new rule for the model, all instances that the rule covers are removed from the instance set, and the remaining instances are considered for the following steps. When a class is covered, the remaining classes are considered, in turn. This process iterates until the instance set becomes empty, that is, all instances are covered.

After formation of the rule set, if the removal of any rule does not change the performance on training set, such rules are removed from the model. Furthermore, to reach an optimum rule set, an optimization procedure is used [57].

The rule induction algorithms for classification, such as Swap-1, can also be applied to regression problems. Since these algorithms are designed for the

prediction of nominal attributes, using a preprocessing procedure, the numeric attribute in regression to be predicted is transformed to a nominal one.

```

[1] Input:  $\{y\}$  a set of output values
[2] Initialize  $n$  = number of cases,  $k$  = number of classes

[3] repeat for each  $Class_i$ 
[4]    $Class_i$  = next  $n/k$  cases from list of sorted  $y$  values
[5] end

[6] repeat for each  $Class_i$  (until no change for any class)
[7]   repeat for each case  $j$  in  $Class_i$ 
[8]     1. Move  $Case_{ij}$  to  $Class_{i-1}$ , compute  $Err_{new}$ 
[9]     If  $Err_{new} > Err_{old}$  return  $Case_{ij}$  to  $C_i$ 
[10]    2. Move  $Case_{ij}$  to  $Class_{i+1}$ , compute  $Err_{new}$ 
[11]    If  $Err_{new} > Err_{old}$  return  $Case_{ij}$  to  $C_i$ 
[12]   next  $Case_j$  in  $Class_i$ 
[13] Next  $Class_i$ 

[14] repeat for each  $Class_i$  (until no change for any class)
[15]   If  $Mean(Class_i) = Mean(Class_j)$  then
[16]   Combine  $Class_i$  and  $Class_j$ 
[17] end

```

Figure 2.5. Composing Pseudo-Classes (P-Class)

For this transformation, the P-class algorithm, shown in Figure 2.5, is used in [59]. This transformation is in fact a one-dimensional clustering of training instances on response variable y , in order to form classes. The purpose is to make y values within one class similar, and across classes dissimilar. The assignment of these values to classes is done in such a way that the distance between each y_i and its class mean must be minimum.

The P-Class algorithm does the following. First it sorts the y values, then assigns an approximately equal number of contiguous sorted y_i to each class. Finally, it moves a y_i to a contiguous class if it reduces the distance of it to the mean of that class.

This procedure is a variation of the *KMEANS* clustering algorithm [16, 35]. Given the number of initial clusters, on randomly decomposed clusters, the

1. Generate a set of Pseudo-classes using the P-Class algorithm.
2. Generate a covering rule-set for the transformed classification problem using a rule induction method such as Swap-1.
3. Initialize the current rule set to be the covering rule set and save it.
4. If the current rule set can be pruned, iteratively do the following:
 - a) Prune the current rule set.
 - b) Optimize the pruned rule set and save it.
 - c) Make this pruned rule set the new current rule set.
5. Use test instances or cross-validation to pick the best of the rule sets.

Figure 2.6. Overview of Method for Learning Regression Rules

KMEANS algorithm swaps the instances between the clusters if it increases a clustering measure or criterion that employs inter and intra-cluster distances. Given the number of classes, P-Class is a quick and precise procedure. However, no idea is stated in the literature about an efficient way to determine the number of classes.

After the formation of classes (pseudo-classes) and the application of a rule induction algorithm to these classes, such as Swap-1, in order to produce an optimum set of regression rules, a pruning and optimization procedure can be applied to these rules, as described in [57, 59]. An overview of the procedure for the induction of regression rules is shown in Figure 2.6.

The naive way to predict the response for a query instance is to assign the average of responses. The average may be a median or mean of that class. However, different approaches also can be considered by applying a parametric or non-parametric model for that specific class. For example, the nearest neighbor approach is used for this purpose, and significant improvements of this combination against the naive approach are reported in [59].

2.4 Projection Pursuit Regression

One problem with most local averaging techniques, such as the nearest-neighbor, is the *curse of dimensionality*. If a given amount of data is distributed in a

high-dimensional space, then the distance between adjacent data points increases with increasing number of dimensions [29]. Friedman and Stuetzle give a numeric example about this problem [20]. Projection pursuit regression (PPR) forms the estimation model by reflecting the training set onto lower dimensional projections as a solution for high dimensional data sets.

Another important characteristic of PPR is its *successive refinement* property. At each step of model construction, the best approximation of the data is selected and added to the model, while removing the well described portion of the instance space. The search on the data set continues for the remaining part and this process iterates by increasing the complexity of the model at each step. The successive refinement concept is applied to regression in a different way here, by subtracting the *smooth* from residuals. A *smooth* is a function formed by averaging responses (y). An example of smooth is shown in Section 2.4.2.

The model approximated by the PPR algorithm is the sum of the smooth functions S of the linear projections, determined in each iteration:

$$\varphi(x) = \sum_{m=1}^M S_{\beta_{\mathbf{m}}}(\beta_{\mathbf{m}} \cdot \mathbf{X}) \quad (2.10)$$

where $\beta_{\mathbf{m}}$ is the parameter vector (projection), \mathbf{X} is the training set against predictor variables, $S_{\beta_{\mathbf{m}}}$ is the smooth function and M is the number of terms or smoothes in the model.

2.4.1 Projection Pursuit Regression Algorithm

At each iteration of the PPR algorithm, a new term, m in Equation 2.10, is added to the regression surface φ . The critical part of the algorithm is the search for the coefficient vector β or projection of the next term. After finding a coefficient vector at each iteration, the smooth of the estimated response values resulting from the inner product ($\beta_{\mathbf{m}} \cdot \mathbf{X}$) is added to the model as a new term, where the term is a function of all features. The linear sum of these functions (2.10) forms the model, which is employed for the prediction task.

- [1] $r_i \leftarrow y_i$, $M \leftarrow 0$, $i = 1, \dots, n$
- [2] Search for the coefficient vector β_M , that maximize fitting criterion $I(\beta)$ by using Equation 2.11
- [3] If $I(\beta)$ is greater than the given threshold
- [4] $r_i \leftarrow r_i - S_{\beta_{M+1}}(\beta_{M+1} \cdot \mathbf{x}_i)$, $i = 1, \dots, n$
- [5] $M \leftarrow M + 1$
- [6] go to *Step 2*
- [7] Otherwise stop, by excluding last term M .

Figure 2.7. Projection Pursuit Regression Algorithm

The search for the coefficient vector for each term is done according to a fitting criterion (figure of merit) such that, the average sum of the squared differences between residuals and the smooth is the minimum. For this purpose, $I(\beta)$, the fraction of unexplained variance that is explained by smooth S_β , is used as an optimality criterion or figure of merit. $I(\beta)$ is computed as

$$I(\beta) = 1 - \frac{\sum_{i=1}^n (r_i - S_\beta(\beta \cdot \mathbf{x}_i))^2}{\sum_{i=1}^n r_i^2} \quad (2.11)$$

where r_i is a residual which takes the value of y_i in the first step of the algorithm. The coefficient vector β that maximizes $I(\beta)$ is the optimal solution.

In the first line of the algorithm current residuals and the term counter are initialized. In the second step, the coefficient vector that results in the best smooth close to the residuals according to the fitting criterion I is found. A smooth is found for each β vector, in ascending order of the linear combination $(\beta \cdot \mathbf{X})$. If the criterion value found is below a given threshold, the iteration of the algorithm is continued by the new residual vector, which is found by subtracting the smooth from the current residuals at *Step 4*. With this subtraction operation, the algorithm gains the successive refinement characteristic.

For search of the coefficient vector that maximizes the fitting criterion, a modification of the Rosenbrock method [50] is chosen in [20], and as a smoothing procedure, a method is described in the next subsection.

Some models approximate the regression as a sum of the functions of individual predictors (standard additive models), and because of that, they can

not deal with interactions between predictors. In such models, the projections are done onto individual predictors rather than onto a projection vector, which is the linear sum of the predictors, as in PPR. These projection vectors, instead of individual predictors, allow PPR to deal with interactions, which is the third main property of PPR.

2.4.2 Smoothing Algorithm

Traditional smoothing procedures assume that the observed variation, response y_i , is generated by a function which has a normally distributed error component. The smooth constitutes an estimation for that function. As an example, in simple linear regression, this function is a linear combination of predictors. As stated above, PPR tries to explain this variation with not just one smooth, but with a sum of smoothes over linear combinations of predictors.

Generally, the smooth functions employed here are not expressions, rather, they are a local averaging of the responses or residuals. Taking the averages of responses in neighborhood regions forms this smooth function. The boundaries of the neighborhood region where the averages are taken are called *bandwidth*. For example, in the *k-nearest neighbor* algorithm, k is used for the constant bandwidth. In [20], a variable bandwidth algorithm is employed, where larger bandwidths are used in regions of high local variability of response. To clarify the concept of smoothing, we describe the constant bandwidth smoothing algorithm of Tukey [52] called “running Medians”.

Running medians is a simple procedure that averages the response by taking the median of the neighbor region. *Running medians of three* algorithms, described in [52], are shown with a simple example in Figure 2.8. The smooth of each response is found by the median of three values in the sequence. One of them is the response itself, and other two are neighbors.

<i>Given</i>	:	4	7	9	3	4	11	12	1304	10	15	12	13	17
<i>Smooth</i>	:	?	7	7	4	4	11	12	12	15	12	13	13	?

Figure 2.8. Running Medians of Three

Friedman and Stuetzle [20], employ *running medians of three* in their variable bandwidth smoothing algorithm, as is shown in Figure 2.9.

- [1] Running medians of three;
- [2] Estimating the response variability at each point by the average squared residual of a locally linear fit with constant bandwidth;
- [3] Smoothing this variance estimates by a fixed bandwidth moving average;
- [4] Smoothing the sequence obtained by pass (1) by locally linear fits with bandwidths determined by the smoothed local variance estimates obtained in pass (3).

Figure 2.9. Variable Bandwidth Smoothing Algorithm

In *Step 1*, a smooth for the response is formed. In *Step 2*, for each smoothed response value, we find the variance of the neighbors in the interval determined by a given constant bandwidth. In *Step 3*, these variances are smoothed by a given constant bandwidth. Finally, by employing these smoothed variance values as a bandwidth for each smoothed response determined in *Step 1*, a variable bandwidth smooth is obtained.

2.5 Regression by Tree Induction

Tree induction algorithms construct the model by recursively partitioning the data set. The task of constructing a tree is accomplished by employing a search for an attribute to be used for partitioning the data at each node of the tree. The explanation capability of regression trees and their use to determine key features from a large feature set are their major advantages. In terms of performance and accuracy, regression tree applications are comparable to other models. Regression trees are also shown to be strong when there are higher order dependencies among the predictors.

One characteristic common to all regression tree methods is that, they partition the training set into disjoint regions recursively, where the final partition is determined by the leaf nodes of the regression tree. To avoid overfitting and form simpler models, pruning strategies are employed in all regression tree methods.

In the following subsections, three different regression tree methods are described: CART, RETIS and M5. They share the common properties described above, but show significant differences in some of measures and traits they demonstrate.

2.5.1 CART

Using trees as regression models was first applied in the CART (Classification and Regression Trees) program, developed by the statistical research community [9]. This program induces both regression and classification trees.

In the first step, we start with the whole training set represented by the root node to construct the tree. A search is done on the features to construct the remaining part of the tree recursively. We find the best feature and feature value of any instance at which to split the training set represented by the root node. This splitting forms two leaf nodes that represent two disjoint regions in the training set. In the second step, one of these regions is selected for further splitting. This splitting is again done according to a selected feature value of an instance. At each step of partitioning, one of the regions, which are not selected before are taken and partitioned to two regions in the same manner along a feature dimension.

After forming regions, which are represented by the leaf nodes of a tree, a constant response value is used for estimation of a query. When a test instance is queried, the leaf node that covers the query location is determined. A constant average value of response values of the instances of the region is assigned as the prediction for the test instance. Each disjoint region has its own estimated value that is assigned to any query instance located in this region.

To construct optimum disjoint regions, an error criterion is used. The optimum value of this criterion produces a decomposition at any step of the tree induction process described above so that the correct region, feature, feature value (splitting surface) and estimates for each region are selected. To determine the predicted target values in these regions, averaging methods such as mean and median are used. As a fitting criterion, the variances of the regions

are used (2.13).

$$Error(Variance) = \sum_{i=1}^n (y_i - \bar{y})^2 \quad (2.12)$$

where n is the number of instances in the region.

$$Splitting\ Error = \frac{1}{n} \left\{ \sum_{\mathbf{x}_i \in \mathbf{X}_{left}} (y_i - \bar{y}_{left})^2 + \sum_{\mathbf{x}_j \in \mathbf{X}_{right}} (y_j - \bar{y}_{right})^2 \right\} \quad (2.13)$$

After computing the splitting error for all possible splits of a particular predictor, the splitting that maximizes the following criterion is selected.

$$C = Variance - Splitting\ Error \quad (2.14)$$

The node and predictor that reach the maximum criterion C , are selected for splitting. An example regression tree is shown in Figure 2.10. The construction process is illustrated in Figure 2.11.

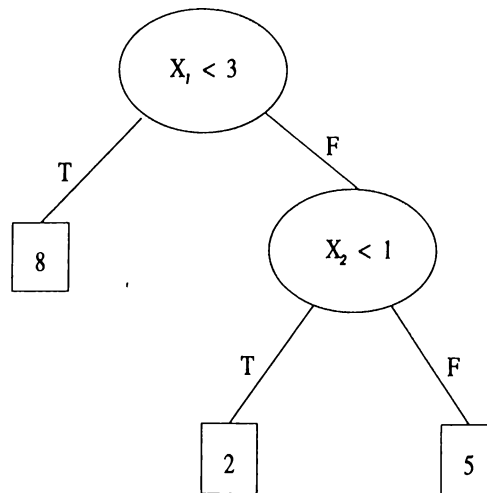


Figure 2.10. Example of Regression Tree

Formally, the resulting model can be defined in the following form [9, 19]:

$$\text{If } \mathbf{x} \in R_m, \text{ then } f(\mathbf{x}) = g_m(\mathbf{x}\{a_j\}_1^P). \quad (2.15)$$

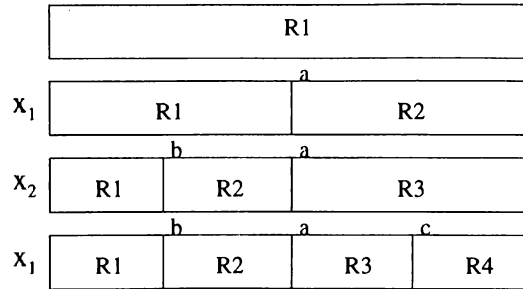


Figure 2.11. An example of the tree construction of process. Four regions are determined by predictors x_1 and x_2 .

where $\{R_m\}_1^P$ are disjoint subregions representing p partitions of the training set. The functions g are generally in simple parametric form. The most common parametric form is a constant function (2.16), which is illustrated with the example given in Figure 2.10.

$$g_m(\mathbf{x}|a_m) = a_m. \quad (2.16)$$

The constant values of leaves or partitions are generally determined by averaging. More formally, the model can be denoted by using basis functions:

$$\bar{f}(\mathbf{x}) = \sum_{m=1}^M a_m B_m(\mathbf{x}) \quad (2.17)$$

The basis functions $B_m(x)$ take the form

$$B_m(\mathbf{x}) = I(\mathbf{x} \in R_m) \quad (2.18)$$

where I is an indicator function having the value one if its argument is true and zero otherwise. Let $H[\eta]$ be a step function, indicating a positive argument

$$H[\eta] = \begin{cases} 1 & \text{if } \eta > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.19)$$

and let $\text{LOF}(g)$ be a procedure that computes the *lack of fit* of an estimation function g to the data. The recursive partitioning algorithm is given in Figure 2.12.

```

[1]  $B_1(\mathbf{x}) \leftarrow 1$ 
[2] For  $M = 2$  to  $M_{max}$  do :  $lof^* \leftarrow \infty$ 
[3]   For  $m = 1$  to  $M - 1$  do :
[4]     For  $v = 1$  to  $n$  do :
[5]       For  $t \in \{x_{vj} | B_m(\mathbf{x}_j) > 0\}$ 
[6]          $g \leftarrow \sum_{i \neq m} a_i B_i(\mathbf{x}) + a_m B_m(\mathbf{x}) H[+(x_v - t)] + a_M B_m(\mathbf{x}) H[-(x_v - t)]$ 
[7]          $lof \leftarrow \min_{a_1 \dots a_M} LOF(g)$ 
[8]         if  $lof < lof^*$  , then  $lof^* \leftarrow lof$ ;  $m^* \leftarrow m$ ;  $v^* \leftarrow v$ ;  $t^* \leftarrow t$  end if
[9]       end for
[10]    end for
[11]  end for
[12]  $B_M(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[-(x_{v^*} - t^*)]$ 
[13]  $B_{m^*}(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[+(x_{v^*} - t^*)]$ 
[14] end for

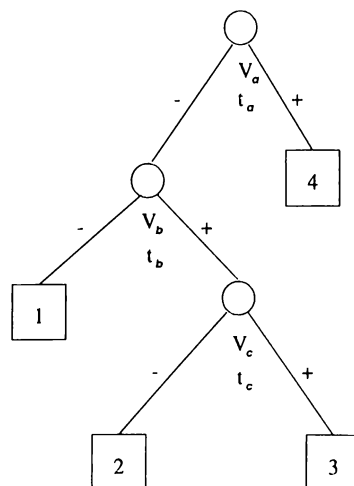
```

Figure 2.12. Recursive Partitioning Algorithm

The first line of the algorithm assigns the whole training set as the initial region. The first loop iterates the splitting until reaching a maximum number of regions. The next three loops selects the optimum basis function B_{m^*} (intuitively the optimum region), predictor x_{v^*} , and split point t^* . At lines 12 and 13, the selected region for splitting, B_{m^*} , is replaced with its two partitions. This is done by adding a factor to its product; with $H[-(x_{v^*} - t^*)]$ for the negative portion of the region at line 12 by creating a new basis function, and with $H[+(x_{v^*} - t^*)]$ for the positive portion of the region at line 13, by modifying or removing the previous basis function. Finally the basis functions formed by the algorithm will take the following form:

$$B_m(\mathbf{x}) = \prod_{k=1}^{K_m} H[s_{km} \cdot (x_{v(k,m)} - t_{km})] \quad (2.20)$$

where the quantity K_m is the number of splits that gave rise to B_m , and the arguments of the step functions contain the parameters associated with each of these splits. The quantity s_{km} takes $(+/-)1$ values indicating the right/left portions, $v(k,m)$ label the predictor variables, and t_{km} represent values on the corresponding variables. A possible output of the algorithm is shown in Figure 2.13.



$$\begin{aligned}
 B_1 &= H[-(x_{va} - t_a)]H[-(x_{vb} - t_b)] \\
 B_2 &= H[-(x_{va} - t_a)]H[+(x_{vb} - t_b)]H[-(x_{vc} - t_c)] \\
 B_3 &= H[-(x_{va} - t_a)]H[+(x_{vb} - t_b)]H[+(x_{vc} - t_c)] \\
 B_4 &= H[+(x_{va} - t_a)]
 \end{aligned}$$

Figure 2.13. A binary tree representing a recursive partitioning regression model with the associated basis functions

The partition may lead to very small regions with a large tree. This situation may cause overfitting with unreliable estimates. Stopping the process early may also not produce good results. The solution to this problem is to employ a pruning strategy.

Pruning the regression tree by removing leaves will leave holes, which is an important problem, since we will not be able to give an answer to queries that fall into these regions or holes. That is why the removal of regions is done pairwise, with siblings, by merging them into a single (parent) region. This pruning strategy is described in [9].

Recursive partitioning regression is an adaptive method, one that dynamically adjusts its strategy to take into account the behavior of a particular problem to be solved [19]. For example, recursive partitioning has the ability to exploit low local dimensionality of functions. In local regions, the dependence of the response may be strong on a few of the predictors, and these few variables may be different in different regions. Another property of recursive partitioning regression is that they allow interpretations, especially when a

constant estimation is done on the leaves.

On the other hand, it has some drawbacks and limitations, the most important is the fact that the estimation is discontinuous. The model cannot approximate even simple continuous functions such as linear functions, which limits the accuracy of the model. As a consequence of this limitation, one cannot extract from the representation of the model the structure of the function (e.g. linear or additive), or whether it involves a complex interaction among the variables.

2.5.2 RETIS

In the basic CART algorithm described above, the estimated response value, \bar{y} on the leaves of the regression tree was a constant function(2.16). On the other hand, RETIS (Regression Tree Induction System) [33, 34], a different approach used to construct regression trees, developed by the machine learning community, is an extension of CART that employs a function on the leaves. This is a linear function of continuous predictors. The use of linear regression at the leaves of a regression tree is called *local linear regression* [33]. RETIS can also be categorized as a LWR system (Section 2.2).

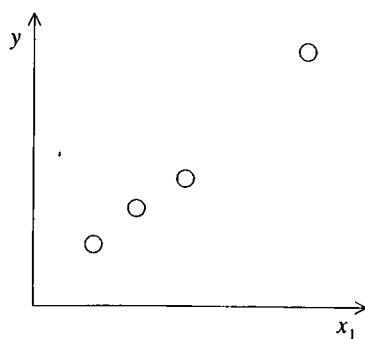


Figure 2.14. An example region, with large variance, which is inappropriate for splitting

RETIS is not just a modification of CART at the leaf nodes. The employment of linear regression enforces modifications in the construction of the regression tree. In the process of tree construction, the CART system forms subtrees to minimize the expected variance (2.13). However, when applying

local linear regression to a regression tree, the variance is not an appropriate measure as an optimality criterion. If the relationship between the predictors and response is linear, this region may not be appropriate for splitting even if the variance is very large. This situation is illustrated with an example in [33]. Suppose we have a region with four instances described with only one predictor as shown in Figure 2.14. Even the error is large in terms of variance, it is almost zero according to a linear approximation on these four points. Such regions are not appropriate for further splitting. That is why an alternative splitting criterion is employed in RETIS as given in Equation 2.22. Let us first define *impurity measure*, I :

$$I(\mathbf{X}) = \sum_{i=1}^n (y_i - g(\mathbf{x}_i))^2 \quad (2.21)$$

where n is the number of instances, g is the linear function that best fits the instances of the region. Consequently, the figure of merit (the splitting criterion) is defined as in Equation 2.22.

$$C = \frac{1}{n} [n_{left} I_{left} + n_{right} I_{right}] \quad (2.22)$$

The use of Equation 2.21 instead of Equation 2.13 in computing figure of merit is the main difference between CART and RETIS. When estimating a response value for a query, the value that results from the linear function on which the leaf node the query falls is used.

After construction of a regression tree, a pruning strategy is employed, as in most other tree induction models. See [41] for an in-depth explanation of pruning. The strategy used in RETIS computes two different error measures: *static error* and the *backed-up error*. The static error is computed at a node, supposing it is a leaf, and backed-up error is computed at the same node for the case, in which the subtree is not pruned. If the static error is less than or equal to the backed-up error, then the subtree is pruned at that node, and the tree node is converted into a leaf node.

2.5.3 M5

M5 is another system [45] that builds tree-based models for the regression task, similar to CART and RETIS. Although the tree construction in M5 is similar to CART, the advantage of M5 over CART is that the trees generated M5 are generally much smaller than regression trees. Standard deviation is employed as the error criterion in M5, instead of variance as used in CART. The reduction on the error (2.23) on subregions after splitting a region is the measure used to decide on splitting:

$$error = \sigma(\mathbf{X}) - \sum_i \frac{|\mathbf{X}_i|}{|\mathbf{X}|} \sigma(\mathbf{X}_i). \quad (2.23)$$

where σ is standard deviation and i is the number of subregions of a region whose instances are denoted by \mathbf{X} . After examining all possible splits, M5 chooses the one that maximizes the expected error reduction (2.23).

M5 is also similar to RETIS in that it employs a linear regression model on the nodes to estimate responses by using standard linear regression techniques [43]. These linear models are constructed on all the nodes, starting from the root down to the leaves. However, instead of using all the attributes or predictors, a model at a node is restricted to the attributes referenced by linear models in the subtree of that node.

After constructing the tree and forming linear models at the nodes as described above, each model is simplified by eliminating parameters to maximize its accuracy. The elimination of parameters generally causes an increase in the average residual. To obtain linear models with fewer of parameters, the value is multiplied by $(n + p)(n - p)$, where n is the number of instances and p is the number of parameters in the model. The effect is to increase the estimated error of models with many parameters and with a small number of instances or training cases. M5 uses a greedy search to remove variables that contribute little to the model. In some cases, M5 removes all of the variables, leaving only a constant [33].

The pruning process in M5 is the same as RETIS. To prune the constructed tree, each non-leaf node is examined, starting near the bottom. If the estimated

error at a node is smaller than its subtree, then that node is pruned.

A smoothing process is employed in M5 for estimation of the response variable. The smoothing process described in [33] is as follows:

1. The predicted value at the leaf is the value computed by the model at that leaf.
2. If the instance follows branch S_i of subtree S , let n_i be the number of training cases at S_i , $PV(S_i)$ the predicted value at S_i , and $M(S)$ the value given by the model at S . The predicted value at S is given by recursive Equation 2.24

$$PV(S) = \frac{n_i PV(S_i) + kM(S)}{n_i + k} \quad (2.24)$$

where k is the smoothing constant.

The accuracy of the model is enhanced by the smoothing process. Improvements in accuracy and model simplification are obtained by M5 over CART, some applications with different training sets are reported in [45]

2.6 Multivariate Adaptive Regression Splines

As stated in the previous section, a fundamental drawback of recursive partitioning regression (CART) is the lack of continuity, which affects the accuracy. Another problem with that method is its inability to provide good approximations to some functions, even to the most simple linear ones. Multivariate adaptive regression splines (MARS) addresses these two problems of recursive partitioning regression, in order to achieve higher accuracy [19].

2.6.1 Piecewise Parametric Fitting and Splines

There are different paradigms for global parametric modeling to generalize low dimensional data. One of them is piecewise parametric fitting. The basic idea is to approximate a function by several simple parametric functions (usually low order polynomials) each defined over different subregions of the training set. The constraint for the formation of polynomial fitting is that it must be continuous at every point.

The most popular piecewise polynomial fitting procedures are based on *splines*, where the parametric functions are polynomials of degree q . The procedure is implemented by constructing a set of globally defined basis functions. These functions span the space of the q th order spline approximations, and fit the coefficients of the basis function to the data using the least squares technique. The spline basis functions are denoted by,

$$\{(x - t_k)_+^q\}_1^K \quad (2.25)$$

where $\{t_k\}_1^K$ is the set of split (knot) locations. The subscript $+$ indicates a value of zero for negative values of the argument. This is known as a truncated power basis in the mathematical literature. A general review of splines is given in [12].

2.6.2 MARS Algorithm

The MARS algorithm is a modified recursive partitioning algorithm, given in the previous section, which addresses the problems stated above. The reason that recursive partitioning algorithms are discontinuous, the first problem, is the use of the step function. If the step function were replaced everywhere by a continuous function where it appears in that algorithm (lines 6, 12 and 13), it could produce a continuous model. The step function employed in that algorithm can be considered as a special case of a spline basis function, where $q = 0$.

The one-sided truncated power basis functions for representing q th order

splines are

$$b_q(x - t) = (x - t)_+^q \quad (2.26)$$

where t is the knot location, q is the order of the spline, and the subscript indicates the positive part of the argument. For $q > 0$, the spline approximation is continuous. A two-sided truncated power basis is of the form

$$b_q^\pm(x - t) = [\pm(x - t)]_+^q \quad (2.27)$$

The step functions that appear in recursive partitioning algorithms are seen to be two-sided truncated power basis functions for $q = 0$ splines. The solution for discontinuity is solved by employing spline functions, of the order of $q > 0$, instead of step functions in the algorithm.

The second modification is related to the second problem, the inability of the algorithm to provide good approximations to certain functions. After the first modification, the algorithm tends to involve functions with more than a few variables (higher order interactions). At each split, one such function is removed, and two new functions are produced with one more variable. This causes a one level increase in the interaction order. With such complex functions, having high level orders, it becomes difficult to approximate simple functions like linear ones.

The solution for this problem is not to delete the lower order parent after splitting. With this modification, all basis functions now become eligible for further splitting. The new model involves either high or low order interactions, or both.

A third problem emerges after the employment of splines in the algorithm. Since the algorithm allows multiple splits on the same predictor, along a single path of the binary tree, final basis functions may include several factors, involving the same variable in their product. For $q > 0$, higher orders than q may be produced on a single predictor.

After the second modification, not deleting the parents after splits, a restriction on the basis function can be applied to involve distinct predictors. Since we do not remove the parent after splitting, many such splits can be

done on the same parent. By employing another split to that parent instead of splitting a child, MARS does not increase the depth or add a new factor to the product.

One remaining problem, which is not solved with MARS, is the value of q . The general idea is to use $q = 1$. A discussion of this problem is given in [19].

In summary, the following modifications are done to the recursive partitioning algorithm: (a) Replacing the step function $H[\pm(x - t)]$ by a truncated power basis function $[\pm(x - t)]_+^q$; (b) not removing the parent basis function B_{m^*} after its split, thereby making it and both its daughters eligible for further splitting; (c) restricting the product associated with each basis function to factors involving distinct predictor variables.

After using two-sided truncated power basis functions, instead of a step function, the MARS algorithm (shown in Figure 2.15), now produces multivariate spline basis functions of the following form:

$$B_m^{(q)}(\mathbf{x}) = \prod_{k=1}^{K_m} H[s_{km} \cdot (x_{v(k,m)} - t_{km})]_+^q \quad (2.28)$$

For pruning of the resulting model after the MARS algorithm, it is now no longer necessary to employ the two-at-a-time deletion strategy used in the previous algorithm. Because the parents are not deleted thus, there will be no holes left after any deletion. Any pruning algorithm can be employed for the MARS procedure.

In the algorithm above, truncated power basis functions ($q = 1$) are substituted for step functions in lines 6, 12 and 13. The parent basis function is included in the modified model in line 6 and remains in the model through lines 12-14. Basis function products are constrained to contain factors involving distinct variables by the control loop in line 4. Figure 2.16 illustrates the regions after constructing the model. Note that the split regions are not deleted from the model, as in CART, and another splitting for the same region can be applied with the same or a different predictor.

```

[1]  $B_1(\mathbf{x}) \leftarrow 1; M = 2$ 
[2] Loop until  $M > M_{max} : lof^* \leftarrow \infty$ 
[3]   For  $m = 1$  to  $M - 1$  do :
[4]     For  $v \notin \{v(k, m) | 1 \leq k \leq K_m\}$ 
[5]       For  $t \in \{x_{vj} | B_m(x_j) > 0\}$ 
[6]          $g \leftarrow \sum_{i=1}^{m-1} a_i B_i(\mathbf{x}) + a_m B_m(\mathbf{x}) H[+(x_v - t)]_+ +$ 
            $a_M B_m(\mathbf{x}) H[-(x_v - t)]_+$ 
[7]          $lof \leftarrow \min_{a_1 \dots a_{M-1}} LOF(g)$ 
[8]         if  $lof < lof^*$ , then  $lof^* \leftarrow lof; m^* \leftarrow m; v^* \leftarrow v; t^* \leftarrow t$  end if
[9]       end for
[10]    end for
[11]  end for
[12]  $B_M(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[+(x_{v^*} - t^*)]_-$ 
[13]  $B_{M+1}(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[-(x_{v^*} - t^*)]_-$ 
[14]  $M \leftarrow M + 2$ 
[15] end loop
[16] end algorithm

```

Figure 2.15. MARS Algorithm

2.7 Discussion

We have reviewed six different regression techniques, each having different characteristics when compared to others. Three of them (instance-based regression, locally weighted regression and rule-based regression) have been developed mainly by the machine learning community, and others (projection pursuit regression, regression tree induction, and multivariate adaptive regression splines) mainly by the statistics community. The common property of all

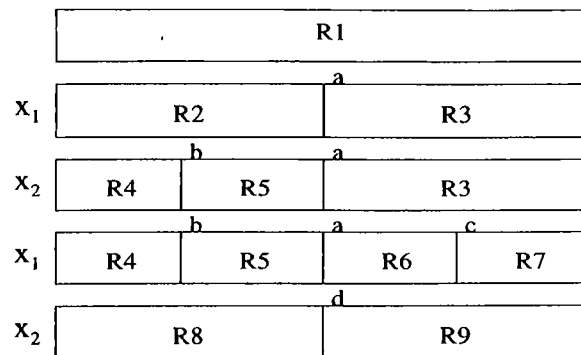


Figure 2.16. An example for the regions of MARS algorithm

these methods is that all of them are non-parametric, and they are the most popular among current regression methods.

In instance-based learning, a lazy approach is employed, where no model is constructed in the training phase. The model is the training set itself. The whole computational complexity of this method is in its prediction, especially the determination of neighbor instances. The prediction is based on the location of the query, and it is computed according to the target values of neighbor instances. The criterion used to detect neighbor instances is the similarity measure based on distance.

Locally weighted regression is another lazy (or memory-based) approach, where the instances are simply stored in memory during the training phase. The difference between locally weighted regression and instance-based methods is in the prediction phase, where a local parametric model is constructed for each query instance by using the neighbor instances. Since, at each query instance, a new local model is constructed, it is more complex than the previous approach.

The projection pursuit regression method has the ability to reduce dimensionality by projecting instances to lower dimensional (one or two) vectors or surfaces. The idea of projection is also used in exploratory data analysis to determine clusters on projections [21]. The same idea is adapted to regression. Successive refinement technique is also applied in the projection pursuit regression, which shows significant improvements for most applications.

All the remaining methods estimate models by partitioning the training set into regions. Rule-based regression techniques accomplish this by partitioning the data using the rule induction techniques of machine learning. On the other hand, in the other partitioning methods (CART, RETIS, M5 and MARS), this is done by splicing the features recursively into two regions, by constructing a binary regression tree. The main difference between these methods and MARS is that MARS is continuous at the borders of the partitioned regions, while others are discrete. CART simply uses the averages of the regions for prediction; RETIS and M5 make prediction by constructing linear models. On the other hand, since MARS produces a large number of overlapping regions,

its computational complexity is larger than other partitioning methods.

Properties	Instance Based Reg. (KNN)	Locally Weighted Reg. (LOESS)	Proj. Pursuit Reg. (PPR)	Rule Based Reg. (Rule)	Tree Based Reg. (CART)	Adaptive Reg. Splines (MARS)
Memory-based	✓	✓				
Partitioning				✓	✓	✓
Interpretable			✓	✓	✓	✓
Adaptive		✓	✓	✓	✓	✓
Incremental	✓	✓				

Table 2.2. Properties of Regression Algorithms (the names of programs developed with those methods are shown in parentheses):

The properties of regression methods are summarized in Table 2.2. Five different properties are used to compare the algorithms. The main characteristic of *memory-based* models is storing the instances and delaying processing to the prediction phase. The model constructed is in fact the training set itself. *Recursive partitioning* algorithms construct the models by partitioning the data into regions. *Interpretability* is one of the main concerns for most knowledge acquisition and knowledge engineering applications, in order to extract information that can be verified by experts. The algorithms covered in this chapter that induce models have this property. If the locations of the test or query instances affect the model, prediction and contribution of variables in the regression task, such algorithms are called *adaptive*. Another important property given in the table is *incremental property* of the algorithm. This is the inverse of batch processing. For large training sets, or databases, particularly processing can be done without loading all of the data set into memory if this property is satisfied. The order of the training instances is ignored when constructing any such model.

Chapter 3

Regression by Partitioning Feature Projections

In this chapter we describe the new regression method called Regression by Partitioning Feature Projections (RFPF). RFPF is an instance-based method where most properties are similar to other instance-based methods such that it is a local, memory-based, lazy and distance-based approach. All such properties of RFPF will be described and discussed in detail in the chapter.

In developing this technique, we have incorporated also some advantages of eager approaches, while eliminating most limitations of both eager and lazy methods.

In Chapter 2, previous approaches for regression were described. If the parametric form of the function to be approximated is known, the best solution is to approximate the parameters of the function. For example if function is linear, linear least squares regression can produce accurate results in the following form.

$$\hat{f}(\mathbf{x}_q) = \sum_{j=1}^p \beta_j \cdot x_{qj} + \beta_0 \quad (3.1)$$

here, p is the number of features, \mathbf{x}_q is the query point, x_{qj} is the j th feature value of the query, β_j is the j th parameter of the function and $\hat{f}(\mathbf{x}_q)$ is the

estimated value of the function for the query point \mathbf{x}_{qj} .

However, the assumption that the approximated function is linear is a very strong one and causes large bias error, especially for many real domains. Many modern techniques have been developed, where no assumption is made about the form of the approximated function in order to achieve much better results. Tree and rule induction algorithms of machine learning are such non-parametric approaches.

Additive regression models [30] and feature projection based classification methods of machine learning such as CFP [24] improves the linear parametric form of the (3.1) by replacing the parameters in this equation with non-parametric functions of the following form.

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^p \hat{g}_j(\mathbf{x}_{qj}) \quad (3.2)$$

where \hat{g}_j is the estimation for feature j .

With this form, the assumption that the approximated function is parametric is removed. However, it is assumed that the input features or variables additively form the approximated function. It is shown that for classification tasks of many real world domains, for example that of the data sets used for classification in UCI repository, additive forms achieves high accuracy [31, 24]. Even though regression and classification are similar problems, one predicts a continuous and the other predicts a categorical target, their characteristics are different, and they are investigated independently in the literature. In order to achieve high accuracies in regression problems, interaction effects of features, additional to main (additive) effects, must be handled properly. This is also shown empirically in Chapter 4 by comparing the additive form of RPPF with its original form by using many real world domains obtained from different sources.

There are many approximation techniques that can cope with interaction effects. KNN [40] and partitioning approaches such as rule-based regression [59, 60] tree-based regression [9, 23] and MARS [19] are such techniques. Among projection-based methods, only projection pursuit regression, PPR [20],

handles interactions with the following model.

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^M \hat{g}_m \left(\sum_{j=1}^p \beta_{mj} \cdot \mathbf{x}_{qj} \right) \quad (3.3)$$

where M is the number of projections, β_{mj} is the j th parameter of the m th projection axis and f_m is the smooth or approximation function for m th projection axis.

Here the instances are not projected to feature dimensions. Instead, they are projected to projection axes, found through complex computations [20]. The whole model is constructed with successive M steps, and at each step of the model construction process, a new projection is found which is a linear equation. We think that if there are both interactions and additive (main) effects in a domain, most models that handle interactions, including PPR, may lose some information by not evaluating main effects by using individual features.

RFPF is a projection-based approach that can handle interactions. However, if main effects are higher than interaction effects in a domain, or some features have only main effects, which is probably the case for most real world regression problems, the functional form of RFPF, given below (3.4) enables those effects to be incorporated in the solution properly.

$$\hat{f}(\mathbf{x}_q) = \sum_{R' \in \{R_s, \mathbf{X}\}} \sum_{j=1}^p \hat{g}_{j,R'}(\mathbf{x}_q) I(j, R') \quad (3.4)$$

where R' is either the whole instance space \mathbf{X} or the region obtained after s partitioning steps, R_s ; and $I(j)$ is an indicator function whose value is either 0 or 1, according to the feature j .

RFPF incorporates interactions as partitioning techniques do, by partitioning the instance space. However, this partitioning does not produce disjoint regions, such as in C4.5 for classification and CART for regression. Instead, these are overlapping regions similar to MARS, DART and KNN. Query instances are always close to the center of these regions, which is the way nearly all lazy approaches work. If some features do not have interactions with others,

which is the situation for most cases, RFPF incorporate main effects of these features as much as possible by using the whole instance space, with much crowded instances as additive methods. It decreases the effects of curse of dimensionality, a problem for almost all other approximation techniques except projection-based approaches. On the other hand, if a feature has interactions with others, the region after partitioning, R_s , is used instead.

3.1 RFPF Algorithm

The main property of RFPF is that, a different approximation is done for each feature by using the projections of the training instances on each feature dimension separately. These approximations may be different for each feature and for each query point. A partitioning strategy is employed in the algorithm and some portion of the data is removed from the instance space at each step. The same approximations are repeated for a sequence of partitioning steps, where it continues until reaching a small number of instances.

For all query instances the procedure described above is applied. This produces different regions and different contribution of features for each query in the instance space, which enables the context-sensitive solutions.

3.1.1 Training

Training involves simply storing the training set as their projections to the features. This is done by associating a copy of target value with each feature dimension, then sorting the instances for each feature dimension according to their feature values. If there are missing feature values, they are placed at the farthest end of the feature dimensions. These instances, having missing values for the feature dimension, do not effect the results for those features. An example training set with 2 features and 10 training examples projected to these features is shown in Figure 3.1.

f_1	:	2	4	6	8	9	11	14	16	17	18
TARGET	:	14	14.5	16	2	3	3.5	4	8	9	8.5
f_2	:	1	3	4	6	8	20	24	28	32	36
TARGET	:	14	9	3	8.5	2	4	16	8	14.5	3.5

Figure 3.1. An example training set projected to two features: f_1 and f_2 .

3.1.2 Approximation using Feature Projections

In this section, we describe how the individual predictions of features are computed for continuous and categorical features.

3.1.2.1 Continuous Features

Approximation at feature projections is the first stage in the prediction phase of RFPF algorithm. Since the location of the query instance is known, the approximation is done according to this location. At each feature dimension, a separate approximation is obtained by using the value of the query instance for that feature.

Taylor's theorem states that if a region is local enough, any continuous function can be well approximated by a low order polynomial within this region [23]. By determining a different linear equation for each different query value at feature dimensions, we can form the function $g(\mathbf{x}_q)$ in Equation 3.4, even it is complex.

Given the linear equation to be approximated in the following form, Equation 3.5, the classical approach is to approximate coefficients of this equation using the least squares error criterion in Equation 3.6.

$$\hat{y}_{qf} = \beta_{0f} + \beta_{1f}x_{qf} \quad (3.5)$$

$$E_f = \sum_{i=1}^n (y_i - \hat{y}_{if})^2 \quad (3.6)$$

where n is the number of training instances, \hat{y}_{if} is the approximation for query at feature f , and y_i is the actual target value.

We employ the weighted linear least squares approximation for the feature predictions. Similar to the standard linear least squares approach, we find the parameters of (3.5), β_{0f} and β_{1f} for each feature by employing a weight function to the least squares error, in order to determine weighted linear least squares approximation.

$$E_f = \sum_{i=1}^n w_{if}(y_i - \hat{y}_{if})^2 \quad (3.7)$$

and

$$w_{if} = \frac{1}{(x_{if} - x_{qf})^2} \quad (3.8)$$

By taking the derivatives of (3.9) to minimize the error E_f , we find the parameters β_{0f} and β_{1f} for weighted linear least squares approximation.

$$E_f = \sum_{i=1}^n w_{if}(y_i - \beta_{0f} - \beta_{1f}x_{if})^2 \quad (3.9)$$

From $\frac{\partial E}{\partial \beta_{0f}} = 0$

$$\beta_{0f}(\sum_{i=1}^n w_{if}) + \beta_{1f}(\sum_{i=1}^n x_{if}w_{if}) = \sum_{i=1}^n y_i w_{if} \quad (3.10)$$

From $\frac{\partial E}{\partial \beta_{1f}} = 0$

$$\beta_{0f}(\sum_{i=1}^n x_{if}w_{if}) + \beta_{1f}(\sum_{i=1}^n x_{if}^2 w_{if}) = \sum_{i=1}^n x_{if} y_i w_{if} \quad (3.11)$$

By solving the above equations, β_{0f} and β_{1f} are found as follows.

$$\beta_{0f} = \frac{\sum_{i=1}^n y_i w_{if} - \beta_{1f} \sum_{i=1}^n x_{if} w_{if}}{\sum_{i=1}^n w_{if}} \quad (3.12)$$

$$\beta_{1f} = \frac{SP_f}{SSx_f} \quad (3.13)$$

where

$$SP_f = \sum_{i=1}^n x_{if} y_i w_{if} - \frac{(\sum_{i=1}^n x_{if} w_{if})(\sum_{i=1}^n y_i w_{if})}{\sum_{i=1}^n w_{if}} \quad (3.14)$$

and

$$SSx_f = \sum_{i=1}^n x_{if}^2 w_{if} - \frac{(\sum_{i=1}^n x_{if} w_{if})^2}{\sum_{i=1}^n w_{if}} \quad (3.15)$$

To illustrate the feature averaging phase, we can compute a prediction for an example query for the training set given in Figure 3.1. Suppose $f_1 = 12$ and $f_2 = 5$ are feature values of a query instance. Coefficients of the approximated weighted linear least squares equations for these features and the feature predictions are shown in Figure 3.2.

β_{0f_1}	: 5.037	β_{0f_2}	: 6.779
β_{1f_1}	: -0.034	β_{1f_2}	: -0.091
\bar{y}_{f_1}	: 4.630	\bar{y}_{f_2}	: 6.320

Figure 3.2. Approximations for Feature Projections

3.1.2.2 Categorical Features

The weighted linear least squares approximation is not appropriate for some cases encountered in real life applications. One of them is categorical feature values. Since there is not an ordering between most categorical features extracting a linear relation at any region the query instance fall, is not meaningful. On the other hand, if a categorical feature has an ordering between categorical values (e.g. days of a week), then it can be evaluated by defining it as linear.

Another situation is possible for linear features. If all the instances have same linear value for a particular feature dimension, the slope of the equation will be infinity. This situation can be determined by looking at the value of SSx in Equation 3.15. If $SSx = 0$, we can not employ the weighted linear least squares approximation. This refinement is done to determine such situations together with categorical features.

Those situations can be handled easily by employing an averaging procedure instead of linear approximation. For the RFPF algorithm, mean values of the target values are used as an approximation on such feature dimensions, as shown in Equation 3.16. If none of the values of a categorical feature matches the feature value of the query instance, the contribution of that feature in the final prediction is excluded.

$$\bar{y}_{qf} = \frac{\sum_{i=1}^n y_i}{n} \quad (3.16)$$

3.1.3 Local Weight

Some regions on a feature dimension may produce better approximations when compared to others. In order to obtain a degree of prediction ability of a region on feature dimension, we employ a measure in the prediction algorithm. If the region that query point falls in is smooth, we give a high weight to that feature in the final prediction. By this way we reduce the effect of irrelevant features, as well as the irrelevant regions of a feature dimension. This establishes an adaptive or context sensitive nature, where at different locations in the instance space, the contribution of features on the final approximation differs.

3.1.3.1 Continuous Features

In order to measure the degree of smoothness for continuous features we compute the distance weighted mean squared residuals. Residuals are differences between target values of the instances and their predicted values found by weighted linear least squares approximation for the feature value of each instance. We denote this measure with V_f as given in (3.18). By subtracting it from the variance of the target values of all instances, V_{all} , we find the explained variance according to the region the query instance falls in. By normalizing it with the variance of training set we obtain a measure, called *prediction index* (PI) (3.20). We use the squared PI as the *local weight* (LW) for each feature (3.21).

$$V_{all} = \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n} \quad (3.17)$$

where \bar{y} is the mean of target values of training set.

$$V_f = \frac{\sum_{i=1}^n w'_i (y_i - \beta_0 - \beta_1 x_{if})^2}{\sum_{i=1}^n w'_i} \quad (3.18)$$

where w'_i is defined in Equation (3.19). For an overview of weight functions for regression problems see [7].

$$w'_{if} = \frac{1}{1 + (x_{if} - x_{qf})^2} \quad (3.19)$$

Prediction index of feature f :

$$PI_f = \frac{V_{all} - V_f}{V_{all}} \quad (3.20)$$

Local weight of feature f :

$$LW_f = \begin{cases} PI_f^2 & \text{if } PI_f > 0 \\ 0 & \text{otherwise} \end{cases} \quad (3.21)$$

For the example query on the training data in Figure 3.1 the local weight for f_1 is 0.405, local weight for f_2 is 0.297.

3.1.3.2 Categorical Features

For the computation of local weight for categorical features, a refinement is required. By replacing (3.18) with (3.22), in such a case, we can use the same procedure used for continuous features for the computation of local weight. Note that w'_{if} in (3.19) will be 1 for all the same categorical values.

$$V_f = \frac{\sum_{i=1}^{N_c} w'_i (y_i - \hat{y}_{qf})^2}{\sum_{i=1}^{N_c} w'_i} \quad (3.22)$$

where N_c is the number of instances having the same categorical value, and \hat{y}_{qf} is the average of their target values.

3.1.4 Partitioning Algorithm

Partitioning enables us to deal with interactions among features. If there is no interaction among some features, we use the results we obtained and recorded for these features before the partitioning of the instance space. We try to figure out interactions by looking at local weights before and after partitioning.

Partitioning is an iterative procedure applied to each query instance, where the remaining final region may differ for each query instance. It improves the context-sensitive nature of RFPF such that, the edges of the final region, a hyper-rectangle, are not equal in length for each query, according to the relevancy of features for the prediction of the query. This causes longer edges for less relevant features, and much shorter edges for relevant ones. The region is formed by the partitioning algorithm that will be described in this section, by using an iterative procedure that continues until a small number of instances, say k , are left. This is taken by default 10 in the experiments.

In the first step of partitioning, the predictions and local weights of the features are found and recorded. The feature with the highest local weight for the partitioning of the data is used. Partitioning is done on this feature dimension. The farthest instances to the query value on this feature dimension are marked. The number of these instances are determined by using local weight of that feature, then they are removed on all feature dimensions. If the feature selected for partitioning is nominal, simply all the instances having different nominal values on that feature are also removed. After shrinking the marked instances on all feature projections, partitioning continues by selecting a new feature at each step.

The partitioning algorithm applies a strategy in order to select the right feature for partitioning. For example, if the feature selected in the first step again has the highest local weight for the query in the second step, then the feature having the second highest local weight is selected. By this way, we can pass possible ridges in the data set, so that, selecting a feature with small local weight or that of some others, may increase their local weights in forthcoming steps significantly. However, at a particular step the features with zero local weights are not used for partitioning for that step, unless all local weights

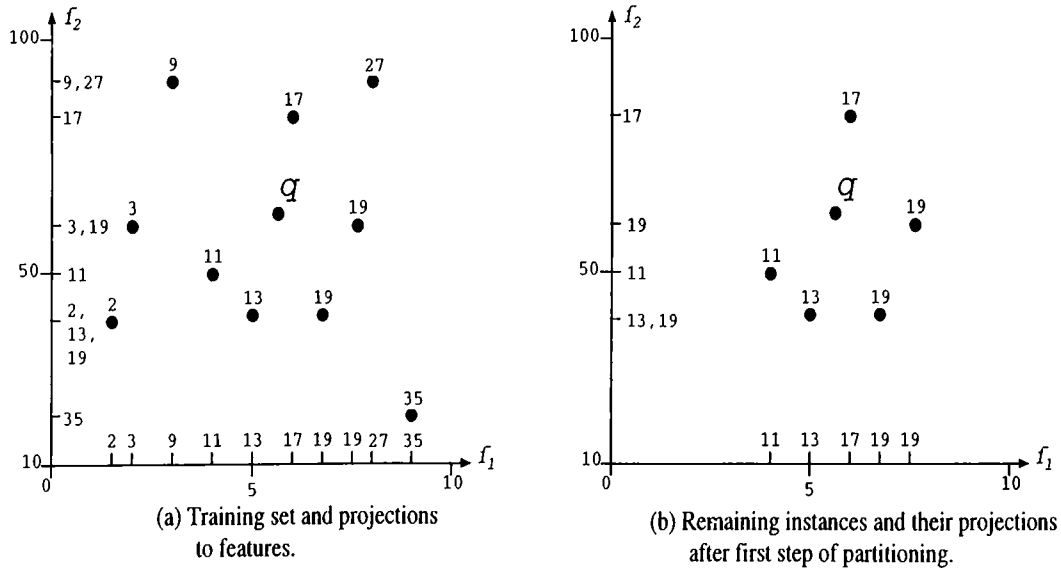


Figure 3.3. Example data set and its partitioning.

in a particular step are zero. This strategy decreases the effect of irrelevant features, especially in high-dimensional domains. Since all the features may have been selected in previous steps, a counter is associated with each feature in order to give chance to different features each time. An example training set and its partitioning on feature f_1 is illustrated in Figure 3.3. In this example, we suppose that local weight of f_1 is 0.5 and k is small enough.

A different strategy is applied for nominal features. If a nominal feature is selected for partitioning once, it is never used again for partitioning. The partitioning algorithm of RFPF is shown in Figure 3.4. The partitioning is repeated for all query instances by using a copy of the feature projections of the data obtained in the training phase.

At line 30, in Figure 3.4, number of steps for the partitioning is recorded to be used in the final prediction phase. At line 27, a partitioning of the remaining training set, D' , is employed along the feature dimension, $MaxF$, selected for partitioning.

Suppose, at any particular step of partitioning, the best feature along with the partitioning will occur has been found according to the partitioning strategy. We must determine the number of instances that will remain after partitioning, n' , according to the local weight of the feature in that step. However,


```

[1]  $n' \leftarrow n; S_{max} \leftarrow \log n; s \leftarrow 0; D' \leftarrow D$ 
[2] For  $f = 1$  to  $p$ 
[3]    $priority(f) \leftarrow S_{max}$ 
[4] end for

[5] While  $n' > k$  and  $s < S_{max}$ 
[6]    $s \leftarrow s + 1$ 
[7]   For  $f = 1$  to  $p$ 
[8]     if  $x_{qf}$  is known then
[9]       compute and record  $LW_f(s)$  and  $\bar{y}_{qf}(s)$  on  $D'$ 
[10]    end if
[11]  end for
[12]   $MaxF \leftarrow$  any  $f$  where  $x_{qf}$  is known and  $LW_f(s) > 0$ 
[13]  For  $f = 1$  to  $p$ 
[14]    if  $LW_f(s) > 0$  and  $x_{qf}$  is known then
[15]      if  $priority(f) > priority(MaxF)$  then  $MaxF \leftarrow f$  end if
[16]      if  $priority(f) = priority(MaxF)$  then
[17]        if  $LW_f(s) > LW_{MaxF}(s)$  then  $MaxF \leftarrow f$  end if
[18]      end if
[19]    end if
[20]  end for
[21]  if  $MaxF$  is continuous then
[22]     $priority(MaxF) \leftarrow priority(MaxF) - 1$ 
[23]  end if
[24]  if  $MaxF$  is nominal then
[25]     $priority(MaxF) \leftarrow 0$ 
[26]  end if
[27]   $D' \leftarrow partition(D', MaxF)$ 
[28]   $n' \leftarrow$  size of  $D'$ 
[29] end while
[30]  $S \leftarrow s$ 

```

Figure 3.4. Partitioning Algorithm

if we use local weight as a ratio of removing instances, since it takes values between 0 and 1, all the instances will remain or all of them will be removed for extreme values. The solution to this problem is brought by windowing the local weight to a narrower interval. Its size is determined by a windowing constant, c_w , that takes values between 0 and 0.5, leading a local weight interval, $[0.5 \mp c_w]$. Local weights are transformed to this interval. Thus for $c_w = 0.3$, the value we have used in experiments, the largest local weight becomes $LW_{max} = 0.8$ and smallest one becomes $LW_{min} = 0.3$ after this transformation. The equation used for transformation is given below (3.23).

$$n_a = (n_b - m_f)(LW_{max} - (LW_{max} - LW_{min})LW_f) + m_f \quad (3.23)$$

where n_a and n_b are number of instances after and before partitioning respectively, and m_f is the number of missing values at dimension f .

After determining the number of instances that will remain after partitioning according to the local weight of the selected feature, the farthest instances according to the query value, in the selected feature dimensions are marked until reaching that number of instances are left. The instances having missing values for that feature are excluded from this marking process, and they remain at the end of the feature dimension. If other feature values of such missing valued instances are close to their query values in those dimensions, this enables better accuracy for their predictions, however we always exclude them from the computations in the dimension where their values are missing. Finally feature values of all marked instances are removed from all dimensions. For the example data set, the instances after first step of partitioning according to f_1 , which has higher local weight, is shown in Figure 3.5; and new results are shown in Figure 3.6.

3.1.5 Prediction

The partitioning process results in a region with the query instance in its center. Then we compare local weights obtained for a feature for this region and for the whole region before partitioning. This comparison is performed for each

f_1	:	8	9	11	14	16
TARGET	:	2	3	3.5	4	8
f_2	:	4	8	20	28	36
TARGET	:	3	2	4	8	3.5

Figure 3.5. Example training set after partitioning.

β_{0f_1}	:	-1.759	β_{0f_2}	:	2.868
β_{1f_1}	:	0.476	β_{1f_2}	:	-0.002
LW_{f_1}	:	0.959	LW_{1f_2}	:	0.981
\bar{y}_{f_1}	:	3.950	\bar{y}_{f_2}	:	2.860

Figure 3.6. Approximations for Feature Projections

feature separately. If the local weight of a feature on the initial projections of the instances is larger than that of the projections of the final region, we use the initial computations for prediction and local weight of that feature. Otherwise we use the computations for the final region for that feature in the final prediction.

If a query value for a feature is missing, that feature is not used in the final prediction. Finally a prediction is done for a query instance by computing the weighted average of feature predictions, where weights are the computed local weights. Prediction algorithm is shown in Figure 3.7. For the query in the example above, the solution is:

$$prediction = (0.959 * 3.95 + 0.981 * 2.86) / 1.94 = 3.4.$$

3.2 RFPF-N Algorithm

We have extended the RFPF algorithm to RFPF-N in order to use it with domains with noisy target values. Instance-based algorithms are robust to noisy or extreme input feature values since the query instances will be far from

```

[1]  $Prediction \leftarrow 0; WeightSum \leftarrow 0$ 
[2]  $S, LW$  and  $\bar{y}_q$  are determined by partitioning algorithm.

[3] For feature  $f = 1$  to  $p$ 
[4]   if  $x_{qf}$  is known then
[5]     if  $LW_f(0) > LW_f(S)$  then
[6]        $Prediction \leftarrow Prediction + \bar{y}_{qf}(0)$ 
[7]        $WeightSum \leftarrow WeightSum + LW_f(0)$ 
[8]     end if
[9]   else
[10]     $Prediction \leftarrow Prediction + \bar{y}_{qf}(S)$ 
[11]     $WeightSum \leftarrow WeightSum + LW_f(S)$ 
[12]  end else
[13] end if
[14] end for
[15]  $Prediction \leftarrow Prediction/WeightSum$ 

```

Figure 3.7. Prediction Algorithm

these instances and their effect will be very small [40]. However if the target values of training instances are noisy, this situation must be handled.

We have modified RFPF algorithm, by changing only the feature prediction phase, in order to cope with noisy domains, as described in Section 3.1.2. We have employed an averaging procedure in RFPF-N, instead of weighted linear least squares approximation for feature prediction. This is *distance weighted median* and its algorithm is shown in Figure 3.8, which is used for both categorical and continuous features. For categorical features, the instances which are in the same category as the feature value of query instance are used for computation of both feature prediction and local weight. In the algorithm, equation (3.19) is used as the weight function for feature prediction.

After determining the prediction of a feature, in order to determine its local weight, (3.22) is employed in (3.20), for both categorical and continuous features.

```

[1]  $sum \leftarrow 0$ 
[2]  $weight \leftarrow \sum_{i=1}^{n'} w'_{if}$ 
[3] sort instances according to their target values
[3] While  $sum < weight/2$  take a new instance in the sorted order
[4]    $feature\ prediction \leftarrow y_i$ 
[5]    $sum \leftarrow sum + w'_{if}$ 
[6] end while
[7]  $\hat{y}_{qf} \leftarrow feature\ prediction$ 

```

Figure 3.8. Weighted Median Algorithm

3.3 Properties of RFPF

In this section, we describe important properties and problems for regression algorithms and evaluate RFPF according to them.

3.3.1 Curse of Dimensionality

The curse of dimensionality is a problem for nearly all learning and prediction methods that do not make strong assumptions about the domains. There are some models that handle this situation by making some assumptions. Assumption made in additive models is that features separately contribute to the solutions, as in (3.2). Another solution to this problem comes with projection pursuit regression. The instance space is projected to a lower dimensional space (generally one or two dimensional). However, this approach also has an assumption such that, the information in data can be evaluated by using only the projection of data to some projection axes. Assuming linearity between input features and target in the prediction problems can be seen as a sub-category of additive models by comparing (3.1) and (3.2); and it is a strong assumption that is employed in classical linear regression and linear discriminant analysis, which are parametric models.

The strong assumptions made in prediction tasks cause large bias errors in most domains. This is also what the curse of dimensionality problem causes in other non-parametric learning methods. Therefore, generally the choice is

whether to put up with strong assumptions or with the curse of dimensionality.

Most modern techniques for regression in the literature, such as those described in Chapter 2, are developed in order to obtain better accuracy results by eliminating assumptions employed in classical, generally linear and parametric methods. So developing some measures to decrease the effects of curse of dimensionality is important for modern techniques in order to achieve higher accuracies.

The problem can be illustrated with a simple example. Consider a one dimensional input space, where all instances are uniformly distributed and feature values range from 0 to 1. In this situation half of the feature dimension contains half of the instances. If we add one more feature with the same properties to the instance space, using half of each feature dimension will include 1/4th of the instances. One more feature will decrease this ratio to 1/8, and so on exponentially. Adding new features will cause much sparse instance spaces. In order to keep the same ratio for the number of instances in a region we have to increase the volume of the region exponentially. This is because in high dimensional spaces it is impossible to construct regions that have small size simultaneously in all directions and containing sufficient training data; thus, using large regions for approximation causes large bias errors [23].

$$\frac{\text{size}(R_k)}{\text{size}(R_0)} = \left(\frac{k}{n}\right)^{1/p} \quad (3.24)$$

where k is the number of training instances in region R_k and R_0 is the instance space.

Thus, in high dimensions the size of the region will be close to R_0 even for $k = 1$ in (3.24). Curse of dimensionality is a much more important problem for KNN, when compared to eager methods. Nearly all eager learning approaches (Rule-based learning, tree-based learning and MARS) have some measures to decrease the effect of the curse of dimensionality. The most important one is to properly select the features to be included in the model, and decrease the number of dimensions. This is also the reason for the success of eager approaches against irrelevant features.

Another measure is the adaptive nature of partitioning eager approaches. For example, in a uniformly distributed space, after a normalization process, KNN always has regions with a sphere shape, having the same diameter in all dimensions. However this volume is a hyper-rectangle for most eager approaches rather than a sphere or hypercube, since the edge lengths are determined according to the position of the query in the instance space. Intuitively, important features have smaller edges when compared to unimportant features at that location.

Some solutions similar to these measures are available for KNN, by using external feature selection and feature weighting algorithms before applying it [61]. Feature selection can eliminate irrelevant features and feature weighting can produce elliptic regions instead of spherical ones. However, there is still a problem, that the shape of this elliptic region does not change according to the location of the query in the instance space, which is dynamic in most eager approaches.

On the other hand, the problem of curse of dimensionality is much important for KNN when the task is regression instead of classification. There is an important empirical evidence that KNN can achieve high accuracies in many domains for classification. However this is not the situation for regression [23]. This property of KNN will be discussed in the following sections.

RFPF is a member of instance-based approaches, that are local, memory-based, lazy, non-parametric and do not depend on strong assumptions such as those described above. However, RFPF has some measures to decrease the effect of curse of dimensionality.

In the final prediction phase of RFPF, a subset of features are used in additive form, only for their main effects on the target. The curse of dimensionality does not effect their contributions, since feature prediction is determined only on that single dimension. For remaining features, the effect of curse of dimensionality is not severe. The partitioning algorithm either does not allow irrelevant features to effect partitioning (if their local weights are 0), or their effects are small since a dynamic partitioning occurs according to their local

weights. The partitioning strategy of RPPF forms adaptive regions. According to the position of each query instance, the edge lengths of these regions for each feature dimension may change. For remaining features, predictions are done on these regions.

3.3.2 Bias-variance Trade-off

Following the considerations presented in [22], two important error types collectively effect the success of learning approaches according to the underlying problem they are applied. They are *bias* and *variance* errors, caused by under-fitting and over-fitting respectively on the learning application. A decrease in one of those errors, generally causes an increase on the other. However the behavior of interaction between bias and variance differs according to the algorithm and the domains the algorithms are applied. If we illustrate these error components with an example, large K values in the application of KNN algorithm may cause large bias error, on the other hand, small K values may cause large variance error.

Many factors are effective for these error components. The curse of dimensionality, model complexity, model flexibility, local vs. global approximations, assumptions of the learning approach, noise, missing attribute values, number of features and number of observations in applications are some of those. For example large number of features, small number of training instances, many missing values, large local approximation regions, strong assumptions and simple models are among reasons of bias error. The effect of these issues on RPPF will be discussed in the following sections.

An important result presented in [22] is that for classification tasks the major component of the error is formed by variance, on the other hand, for regression problems the bias error becomes important. This is shown as the main reason for the success of the simple nearest neighbor approach such that it over-performs some sophisticated methods for many classification tasks even though the curse of dimensionality problem of KNN causes strong bias. However, this is not the situation for regression, and the effect of bias error is much important unless the underlying domain includes small number of features or

large number of observations.

In learning problems, this trade-off is unavoidable and RFPF casts its vote for variance by employing many arguments to decrease the bias error. The way for handling bias error caused by the curse of dimensionality is described in the previous section. Besides, it does not make strong assumptions as non-parametric methods. It develops flexible, adaptive and locally weighted approximations in small local projections at each feature dimension for each query instance. All these things may increase the over-fitting, which causes an increase on the variance error. However empirical results show that RFPF is much more successful than KNN, which justifies these claims stated about the behavior of classification and regression for the bias-variance trade-off.

3.3.3 Model Complexity and Occam's Razor

William of Occam's Razor principle states that "Entities should not be multiplied beyond necessity" [14]. This idea has been accepted theoretically in many disciplines including machine learning. Its adaptation on learning approaches states that, simpler models must be preferred to complex ones. Two different versions of this idea are described in [14]. One of them is, "Given two models with the same accuracy, the simpler one must be selected because the simplicity is desirable in itself." Especially, if the interpretation of the induced model is concerned, it is widely accepted. On the other hand, another version states that, "Given two models with the same training-set error, the simpler one should be preferred because it is likely to have lower prediction error on the test set." The well known example for this second interpretation is the pruning applied in some eager learning methods in order to achieve better accuracy in unknown test cases.

The second interpretation has been found inconvenient by many researchers recently and some theoretical and empirical work are published supporting this idea [56]. An overview is given by Domingos [14]. The model complexity issue is strongly related with the considerations presented in previous two sections. It is also possible that complex models can produce better accuracies than simpler ones. That is why the belief for second interpretation may cause to

cease developing new complex techniques that will perform well. Especially with very large databases today, with many instances, avoiding complex models in order to prevent over-fitting may cause information loss and poor accuracies.

RFPF is flexible and complex, such that for different query locations in the instance space, producing infinite number of different local approximation functions on many different domains is possible. If we consider many different feature dimensions having such approximations, RFPF becomes more flexible and complex as the number of dimensions increase. The performance results of RFPF on real data sets confirm those resent worries about the second interpretation of the Occam's Razor principle.

3.3.4 Lazy Learning

Lazy learning methods defer most processing to the prediction time. The training phase includes mainly storing the instances. Since those methods store instances without any generalization in the memory, they are also referred to as *memory-based* methods. On the contrary *eager learning* methods complete most processing in the training phase, before a query is given. These two categories among learning methods is probably the most important in order to differentiate learning methods. Both of them have some advantages and disadvantages in itself.

In eager learning, a single global model is used to fit all the training data. The generalization of the whole data set may improve accuracy, besides such representations generally produce comprehensible models that allow interpretation by humans. An other advantage of eager approaches is their fast prediction time. However if the training instances change dynamically, forming a new model each time new instances are added may be time consuming for such domains, if the method is not incremental. If an eager method is incremental, this time it will be sensitive to the presentation order of the instances.

On the other hand, lazy approaches are useful in dynamic domains, since the main processing is accomplished after the query is given. They are incremental in nature, and it is not order sensitive. Lazy learning enables an other

advantage such that they attempt to fit the training data in the region of the query instance, which may allow better fitting. However if the queries are very frequent, this situation makes lazy approaches time consuming because of long prediction time. Besides, their memory cost is larger. Another limitation of lazy approaches is that generally they are not suitable for interpretation, since they do not produce global models.

Being a lazy approach, RFPF has the properties of lazy approaches explained above. When compared to KNN its training phase includes an extra sort operation on the train data. However, RFPF avoids some tasks required in the training phase of KNN. They are normalization and filling missing values and they will be discussed in the following sections. Incrementality of RFPF can easily be enabled by inserting new instances on sorted feature projections. Extracting relative importance of features and determining features having interactions using RFPF are some interpretation tasks that can be researched.

3.3.5 Local Learning

Local learning is a paradigm devoted to lazy and some eager learning (e.g. recursive partitioning) approaches. It is motivated by the Taylor's theorem which states that if a region is local enough any continuous function can be well approximated by a low order polynomial within it [23]. With this approach the instance space is covered by a set of local regions.

Lazy approaches are those mainly benefit from this paradigm, by locating queries in the center of such overlapping regions. However, recursive partitioning approaches that produce disjoint regions have some trouble that the different low order approximations for each region are not continuous at the boundaries of these regions. The adaptive and flexible determination of the boundaries of these regions is their main advantage that allows local approximations. However, this does not prevent discontinuity. Some solutions are developed for the discontinuous approximations of recursive partitioning approaches by producing overlapping regions instead of disjoint ones [19, 23]. We have compared RFPF with those improved implementations of recursive partitioning methods in the next chapter.

The local learning paradigm is strongly related with the curse of dimensionality, which will effect the size of the regions and the performance of approximations in these regions. The measures taken for RFPF against the curse of dimensionality are described above. The discontinuity problem of some local learning approaches is not encountered for RFPF, since, being a lazy approach, queries are always centered in these overlapping regions.

3.3.6 Irrelevant Features and Dimensionality

The sensitivity to irrelevant features is the most important problem for lazy methods. On the other hand, eager approaches are successful in eliminating the effects of irrelevant features. For example, in recursive partitioning regression (e.g. regression tree induction), the partitioning starts from the most significant feature and continues recursively by employing less relevant ones. It is very likely that most of the irrelevant features will not be used in constructing a regression tree.

The reason that irrelevant features cause problems in lazy learners stems from the distance measure used in those methods. In the nearest neighbor approach for example, nearest instances are determined according to a distance measure in a p dimensional space. This is generally the Euclidean distance. In the computation of distance all features are given equal importance including irrelevant ones. This may cause important instances for a query to go away from the query.

Irrelevant features do not cause any difficulty for RFPF, since distances are computed for each feature separately. Another important advantage of RFPF is that it is highly likely for those features to take lower local weights, since the distribution of target values of nearest instances at any query location will be very close to the distribution of the whole target values in the training set (3.20). RFPF is capable of incorporating all features according to their relevancy on the query instance. If the irrelevant features or the relevance of features changes according to the locations of the instance space, this is handled by RFPF, since it is an adaptive approach.

Eager approaches also handle the adaptivity case by arranging the edge lengths of the regions according to relevancy of feature dimensions in the instance space. However, in contrast to irrelevant features problem, another problem may harm most eager approaches. This is high dimensionality. If the number of features is large when compared to the number training cases, in addition to the curse of dimensionality, it is possible that some features will not be evaluated even they are very relevant. This can be illustrated with regression trees. After a small number of steps in the tree construction process, the number of instances at tree nodes may be exhausted before many relevant features get a chance.

This second problem of high dimensionality is also resolved in RFPF, since all the features are used in the final prediction phase.

3.3.7 Context-sensitive Learning

RFPF is an adaptive or context-sensitive method in the sense that in different locations of the instance space the contribution of the features are different. This property is achieved by two characteristics of RFPF. One of them is the partitioning algorithm. The region formed around the query instance is determined adaptively; different features have different lengths of edges in the final region according to the location of query. The other one is in the local weights. Features may take different local weights according to the location of the query. On the other hand, the local weights of features will be different since different instances will be the neighbors at different feature dimensions. The difference in the neighbors will reduce possible over-fitting, similar to sampling approaches such as boosting [10], which brings an advantage to RFPF. Nearly all eager approaches, in some extent, are context-sensitive, while it is one of the limitations of KNN [15].

3.3.8 Missing Feature Values

It is likely that some feature values may be unknown in the application domain. In relational databases, the most suitable form of databases for the most current learning techniques, the problem occurs frequently because all the records of a table must have the same fields, even if values are inexistent for most records [38]. For example, in a hospital database including many fields for many laboratory tests and medical procedures, only a few of these fields will be filled in for any patient.

Even the importance of handling missing values is accepted in literature, the distortion on the information contained in the data caused by missing values is not exactly prevented in many learning techniques [46, 47, 48]. The most common way to handle missing values is to fill those places with some approximations or some constant values. If missing values are very frequent on some rows or columns, removing these instances or features can also be considered.

The most natural solution for handling missing values is leaving those places empty and not to distort the information in the data. Additive models or feature projection based methods handle missing values in that way, since each feature is evaluated separately. However, their limitation is that they assume all features to have independent effects on the target.

RFPF deals with missing values similar to additive or previous feature projection based models, and also resolve the interactions between features by applying a partitioning process. RFPF achieves this by applying approximations on feature projections using only known values, and in partitioning, for a selected feature dimension along with the partitioning occurs, by keeping missing valued instances of that feature.

3.3.9 Interactions

If some input features have inter-relationship such that the effect of any feature on the target is dependent on one or more different input features, those relations are called as interactions. RFPF produces a local region for each query. Making predictions on those regions enable it to handle interactions and to achieve better accuracy.

On the other hand, some research on the classification methods and real data sets show that, generally the main effects of the features are sufficient to determine the target values [24, 31]. If some features have only main effects on targets, RFPF makes predictions for those features by using the whole instance space instead of local region determined by partitioning, since large number of training instances allow better approximations. Another limitation of some partitioning methods, such as regression tree induction is that the partitioning always occurs with many variables and this causes handling of only high-order interactions. This problem makes it difficult to approximate even some simple forms such as linear functions [19].

Dealing with interactions is an important property for regression methods in order to achieve better accuracies. To illustrate it with a simple example, we can consider predicting the area of a rectangle. Both width and length of a rectangle must be evaluated together since predicting the area by using only one of them is not sufficient.

3.3.10 Feature Projection Based Learning

RFPF is a feature projection based learning approach. The results reported for feature projection based classification methods, CFP [24], KNNFPR [5], COFI [25], FIL [6, 27] and VFI [26], and feature projection based regression method RFP [54] motivated us to develop RFPF. The major distinction between those methods and RFPF is its capability of dealing with interactions.

RFPF inherits most advantages of feature projection based approaches (handling missing values, robustness to irrelevant features etc.), on the other

hand benefits from the advantages of nearest neighbor (nearest instances at any dimension have larger effects on solution when compared to others) and partitioning methods (dealing with interactions, adaptivity and regions with flexible edges are formed by partitioning).

3.3.11 Different Feature Types

Induction methods generally accepts two feature types. One of them is *nominal* features, which take binary or categorical values, the other one is *continuous* features which take numeric or real values. Induction methods either can handle both type of features, or use only on type. If the later is the case, some transformation methods are applied. If a method accepts only continuous features, each nominal value of a feature is replaced with a unique numerical value. On the other hand, if a method use only nominal features, the continuous features are transferred to discrete values generally by employing a clustering procedure. In that case, the range of all possible values of a feature is partitioned into intervals and all the values in each interval is replaced with a unique nominal value. Such procedures may be time consuming, and may cause some information loss. For such reasons, RFPF is developed in order to handle both type of features without any modification.

3.3.12 Noise

Instance based algorithms are generally robust to extreme or noisy input feature values since query locations will not be close to these values [40]. On the other hand, most regression approaches, including KNN, are not robust to *target noise*. The empirical results show that robustness to noise in RFPF is better than some other well known methods. Even though, the robustness of RFPF is better when compared to others, it is unacceptable especially for extremely noisy domains.

A measure of robustness is called the *breakdown* point, which is defined to be the smallest percentage of noisy data that can lead prediction to take

unacceptable large value. The RFPF-N algorithm developed for noisy data, achieves close to the highest possible breakdown value of 50%, since it employs median for approximation on feature projections. On the other hand, in comparison, for classical linear least squares regression method, the breakdown point is only 0%.

3.3.13 Normalization

Learning approaches, such as KNN, that employ distance measures require the normalization of feature values in order to give all features equal contribution in the computation of distance. The wider the range of values for a feature, the higher effect it has in the distance computation. For example, without normalization, a feature which includes values for body weight will cause different nearest neighbors to be determined if it is measured with pound instead of kilogram. On the other hand the weight feature will not effect the computation of distance if there exists an other feature, say population, having values that ranges with millions. RFPF eliminates the need for normalization, since approximations are done on each feature separately.

3.4 Limitations of RFPF

RFPF has two main limitations. The effect of redundant features is a common problem for most inductive algorithms, and lack of interpretation is the common shortcoming of instance-based approaches, which are described below. The limitation caused by rectangular regions, common to most learning approaches, is also mentioned.

3.4.1 Redundant Features

In a database, it is possible that the same information may be repeated in different places. Existence of features that have functional dependencies with each other is such a case. A similar case occurs if some features in the data

are obtained by combining some others. This issue is known in statistical literature as *collinearity*, if such a relation occurs between two features, or as *multicollinearity*, if more than two features have similar relationship.

The effect of redundant features in RFPF is seen on the final prediction phase when merging feature approximations. Redundancy will cause similar features to effect the final prediction more than the other features. Intuitively, if one feature is a copy of another for example, the weight of that information will be duplicated in the final result.

3.4.2 Interpretation

The ininterpretability of the constructed model is an important aspect of eager learning algorithms. The conventional motivation of statistical data analysis is to develop simple compact models that are easy to interpret by human experts. However, the accuracy is the main goal in many applications. That is why recent research has resulted in many complex models, hard to interpret, such as neural networks. KNN is also a common lazy approach that does not produce any model for interpretation. RFPF, as a non-parametric lazy approach which does not construct global models, does not have this property either. However, instead of concept descriptions, some information about the relative importance of the features and interactions between them can be determined with further research.

3.4.3 Rectangular Regions

The partitioning algorithm employed in RFPF partitions the instance space around query point by using a single feature at each step, and the space is partitioned along this feature dimension. This process forms regions as hyper-rectangles. It is not always possible that the instances at any query location will have a rectangular shape parallel to feature dimensions. However, this is the way most partitioning approaches work, and it is generally possible to make good local approximations using instances enclosed by hyper-rectangles.

3.5 Complexity Analysis

Since RFPF is a lazy approach, and stores all instances in the memory, a space proportional to the whole training data is required. Given a data set with n instances and m features this space is proportional to $m.n$. Again, for the training phase, the computational complexity of projecting instances to input features, followed by a sort operation on each feature, is $O(m.n.\log n)$. The computation of variance, $O(n)$, of target values for all training data is also computed in the training phase, and it does not change the above complexity.

Computing the prediction of the target value for a query point starts with making a copy of the projections, which has a complexity of $O(n)$. The computation complexity of local approximation in the first step of partitioning is again $O(n)$. The complexity of computing local weight is also $O(n)$, which is also the total computation complexity at first partitioning step. The partitioning at each step removes, on the average, half of the instances. For the whole partitioning process the total computation for a single feature will be proportional to $2n$ since $n + n/2 + n/4 + \dots \approx 2n$. If we compute the complexity for all features we obtain a complexity proportional to $O(m.n)$, which is equal to the complexity of KNN. If we consider situations for nominal features, this complexity is even slightly shorter than linear features. In the worst case, where a nominal feature has two values (only half of the data is removed), it requires on the average the same complexity. The test times of the algorithms, run on the real data sets also shows that the running test time of RFPF is proportional to KNN.

3.6 Comparisons of Regression Methods

In the previous sections we have described some properties and limitations of RFPF, and made some comparisons with other important approaches in the literature. In this section we summarize such properties and comparisons with different important approaches. Methods included are instance-based regression (KNN [40]), locally weighted regression (LOESS [7]), rule-based regression

(RULE [59]), projection pursuit regression (PPR [20]), partitioning algorithms that induces decision trees (CART [9], DART ([23]) and multivariate adaptive regression splines (MARS [19]). The summary of properties is shown in Table 3.1. One interesting result obtained from the table shows that all partitioning methods (RULE, CART, DART, MARS) except RFPF have similar properties. A detailed overview and comparison of these regression techniques is given in [53].

Properties	RFPF	KNN	LOESS	PPR	RULE	CART	DART	MARS
Adaptive	✓		✓		✓	✓	✓	✓
Continuous	✓	✓	✓	✓			✓	✓
Between Regions								
Different	✓	✓			✓	✓	✓	✓
Feature Types								
Curse of Dimensionality	✓			✓				
Incremental	✓	✓	✓					
Handle	✓	✓	✓	✓	✓	✓	✓	✓
Interactions								
Interpretable				✓	✓	✓	✓	✓
Handle Irrel. Features	✓				✓	✓	✓	✓
Local	✓	✓	✓		✓	✓	✓	✓
Memory Cost				✓	✓	✓	✓	✓
Handle Missing Values	✓							
Robust to Noise	RFPF-N							
No Need for Normalization	✓				✓	✓	✓	✓
Regions Overlap	✓	✓	✓	✓			✓	✓
Partitioning	✓				✓	✓	✓	✓
Occurs								
Testing Cost				✓	✓	✓	✓	✓
Train Cost	✓	✓	✓					

Table 3.1. Properties of Regression Algorithms. The (✓) is used for cases if the corresponding algorithm handles a problem or it is advantageous in a property when compared to others.

Chapter 4

Empirical Evaluations

In this chapter empirical evaluation of RFPF and other important regression methods are presented. Even though, the main purpose is to measure the relative performance of RFPF and to compare it with contemporary regression algorithms, another intension is to present a comparison of those methods on a large number of real domains since it is difficult to find such comparative evaluations in the literature.

The algorithms are properly selected according to some criteria. All of them can handle high dimensional domains and accept both categorical and continuous input features. We did not include LOESS for example, since it does not work with higher dimensions, for more than 6 features. On the other hand, they are successful representatives of different approaches, such as regression tree induction, instance-based learning and rule-based learning. Most of them are recently developed and outperform early developed algorithms within the same approach. Finally, all of them are obtained from shared resources or available in published material.

Those algorithms are KNN (instance-based), the most important one since it belongs to the same category as RFPF, RULE (rule-based learning), DART (regression tree induction), and MARS (spline-based, partitioning regression).

In this chapter, we describe the evaluation methodology commonly used to measure accuracy of regression methods. Later, algorithms and real data

sets are described and empirical results are presented, including the accuracy performance, robustness of algorithms to irrelevant features, missing values and noise, and computation times.

4.1 Performance Measure

The accuracy performance of the regression methods is measured by computing the prediction error of the algorithms. Since the target values are continuous, the absolute difference between prediction and true target value in the test example is used. One common measure is the *mean absolute distance* (MAD) [59, 60]. It is the mean of absolute error found for all test examples.

$$MAD = \frac{\sum_{i=1}^T |y_i - \hat{y}_i|}{T} \quad (4.1)$$

where T is the number of test instances.

MAD values depend on the target values in the given domain. MAD will be higher for a domain with high target values than for a domain with low target values. In order to get a normalized performance measure for all data sets, a modified version of MAD, *relative error* (RE) [59, 60] is used in the experiments. Relative error is the true mean absolute distance normalized by the mean absolute distance from the median.

$$RE = \frac{MAD}{\frac{1}{T} \sum_{i=1}^T |y_i - \text{median}(y)|} \quad (4.2)$$

Performance results in the experiments are reported as the average of relative errors measured by applying 10-fold cross-validation on data sets.

4.2 Algorithms Used in Comparisons

In this section the properties of algorithms used in experiments are briefly described.

4.2.1 RFPF

K is the parameter of RFPF that defines the minimum number of instances allowed for a region determined by the partitioning algorithm; and is set to 10. An other parameter is the *windowing constant*, c_w , that is described in Section 3.1.4, and it is taken as 0.3. RFPF-N algorithm is also used for artificial noisy domains extracted from real data sets, to measure robustness to noise.

4.2.2 KNN

The distance weighted KNN algorithm [40] is used here since it performs better than simple KNN that employs simple averaging. The instances close to the query have larger weights, and these weights are determined by inverse squared distance. The distance measure is Euclidean distance. A normalization on test and train input feature values is applied in order to obtain a value range between 0 and 1. For matching nominal values, the difference is measured as 0, and for the difference between different nominal values on a single dimension 1 is assigned.

Missing values are filled with mean values of the feature if it is continuous, or filled with the most frequent categorical value, if that feature is nominal. K is set to 10 for all experiments.

4.2.3 RULE

The latest rule-based regression implementation, written by Weiss and Indurkha [59, 60] is used in experiments. The program is available in the data mining software kit (DMSK), attached to [60].

4.2.4 DART

It is the latest regression tree induction program developed by Friedman [23]. It avoids limitations of disjoint partitioning, used for other tree-based regression

methods, by producing overlapping regions with increased training cost. In the experiments, the maximum dimension (features) parameter, is increased from 100 to 200, in order to enable experiments for irrelevant features.

4.2.5 MARS

The latest shared version of MARS, mars3.6, is used in experiments, which is developed by Friedman [19]. The highest possible interaction level is enabled and linear spline approximation is set, it generally produces better results than cubic splines on most real data sets.

4.3 Real Data Sets

It is possible to obtain large number of real world data sets for classification, however this is not easy for regression. That is why, data sets used in the experiments are collected mainly from three sources [37, 8, 51]. Properties of all data sets are shown in Table 4.1. Detailed information about these data set is available in Bilkent Function Approximation Repository [28]. In order to save space, they are coded with two letters (e.g., AB for Abalone).

4.4 Accuracy

The relative errors of algorithms on 27 real data sets are shown in Table 4.2. The best results, smallest relative errors, are typed in boldface. RFPF achieves best results in 9 of these data sets. DART and MARS achieves the best in 7 and 6 of these data sets, respectively. In the remaining 6 data sets KNN and RULE achieves better accuracy.

One important result extracted from Table 4.2 is the distribution of relative errors for different data sets. We have computed the variance of errors for each algorithm on all data sets. These variance values show that the performance of RFPF is not effected much for different domains. This is an important

Data Set	Code	Instances	Features (C+N)	Missing values	Target Feature
Abalone	AB	4177	8 (7+1)	None	Rings
Airport	AI	135	4 (4+0)	None	Tons of mail
Auto	AU	398	7 (6+1)	6	Gas consumption
Baseball	BA	337	16 (16+0)	None	Salary
Buying	BU	100	39 (39+0)	27	Husbands buy video
College	CL	236	20 (20+0)	381	Competitiveness
Country	CO	122	20 (20+0)	34	Population
Cpu	CP	209	7 (1+6)	None	CPU performance
Electric	EL	240	12 (10+2)	58	Serum 58
Fat	FA	252	17 (17+0)	None	Body height
Fishcatch	FI	164	7 (6+1)	87	Fish weight
Flare2	FL	1066	10 (0+10)	None	Flare production
Fruitfly	FR	125	4 (3+1)	None	Sleep time
Gss2	GS	1500	43 (43+0)	2918	Income in 1991
Homerun	HO	163	19 (19+0)	None	Run race score
Housing	HU	506	13 (12+1)	None	House prices
Normtemp	NO	130	2 (2+0)	None	Heart rate
Northridge	NR	2929	10 (10+0)	None	Earthquake magnit.
Plastic	PL	1650	2 (2+0)	None	Pressure
Poverty	PO	97	6 (5+1)	6	Death rate
Read	RE	681	25 (24+1)	1097	Reader satisfaction
Schools	SC	62	19 (19+0)	1	Reading score
Servo	SE	167	4 (4+0)	None	Rise time of a servo
Stock	ST	950	9 (9+0)	None	Stock price
Television	TE	40	4 (4+0)	None	People per TV
Usnews	UN	1269	31 (31+0)	7624	Rate of Ph.D.'s
Village	VL	766	32 (29+3)	3986	Number of Sheep

Table 4.1. Characteristics of the data sets used in the empirical evaluations.
C: Continuous, N: Nominal.

result that shows the domain independence characteristic of RPFPP, which is important for large databases today, where data that belong to large number of domains is collected together. Increasing number of different domains in databases is one of the reasons that increase the need for automatic knowledge discovery tools and inductive learning algorithms. On the other hand, when the average relative errors of algorithms on real data sets are compared, RPFPP again achieves the smallest average relative error. Also the variance of RE is the smallest.

The final column of Table 4.2 shows the standard deviation of results of algorithms for all data sets. The standard deviation values are used only to determine small number of data sets to be used for further comparisons of algorithms for noise, irrelevant features and missing values. We have determined a subset of data sets that have similar results for the comparison of algorithms for increasing missing values, irrelevant features and noise. Selected data sets with standard deviation less than 0.07 are typed in last column with bold font. Only in one of these selected data sets RPFPP performs best, by chance.

4.5 Robustness to Irrelevant Features

The performance of algorithms on selected data sets (AU, BU, CP, HU, PL and SC) by adding new irrelevant features are shown in Figure 4.1. From graphs it is seen that, the performance of RPFPP is not effected from irrelevant features in all data sets except PL, by preserving nearly a straight line parallel to the horizontal axis. RULE and MARS are also robust to irrelevant features. It is affected from irrelevant features in PL probably because it is a low dimensional data set, initially having only two input features. Note that, in only one of these data sets (BU), RPFPP performs best initially. Most advantages of RPFPP are generally benefited for high dimensions.

These graphs show that RPFPP is not affected much from irrelevant features. This is the major drawback of KNN, the other lazy algorithm in these comparisons, and this is apparent in the graphs. Robustness of RPFPP to irrelevant features is achieved by the local weights assigned to each feature and by

Data Set	RPFP	KNN	RULE	DART	MARS	StdDev
AB	0.675	0.661	0.899	0.683	0.678	0.101
AI	0.473	0.612	0.744	0.720	0.546	0.115
AU	0.334	0.321	0.451	0.333	0.321	0.056
BA	0.664	0.441	0.668	0.497	0.525	0.102
BU	0.792	0.951	0.944	0.883	0.858	0.066
CL	0.692	0.764	0.290	1.854	0.261	0.646
CO	1.301	1.642	6.307	5.110	1.845	2.300
CP	0.650	0.603	0.678	0.571	0.510	0.066
EL	1.009	1.194	1.528	1.066	1.095	0.207
FA	0.667	0.785	0.820	0.305	0.638	0.204
FI	0.243	0.582	0.258	0.190	0.284	0.155
FL	1.218	2.307	1.792	1.556	1.695	0.397
FR	1.056	1.201	1.558	1.012	1.077	0.222
GS	0.566	0.654	0.218	0.359	0.342	0.177
HO	0.868	0.907	0.890	0.769	0.986	0.078
HU	0.618	0.600	0.641	0.526	0.522	0.054
NO	0.962	1.232	1.250	1.012	1.112	0.128
NR	0.947	1.034	1.217	0.928	0.873	0.134
PL	0.415	0.475	0.477	0.404	0.432	0.034
PO	0.703	0.796	0.916	1.251	0.677	0.233
RE	1.008	1.062	1.352	1.045	1.194	0.142
SC	0.319	0.388	0.341	0.223	0.350	0.062
SE	0.527	0.619	0.229	0.441	0.337	0.153
ST	0.729	0.599	0.906	0.781	0.754	0.110
TE	1.659	1.895	4.195	7.203	2.690	2.281
UN	0.666	0.480	0.550	0.412	0.444	0.101
VL	0.970	1.017	1.267	1.138	1.131	0.116
Mean	0.768	0.882	1.162	1.158	0.821	
Variance	0.102	0.220	1.659	2.323	0.310	

Table 4.2. Relative Errors of Algorithms. Best results are typed with bold font.

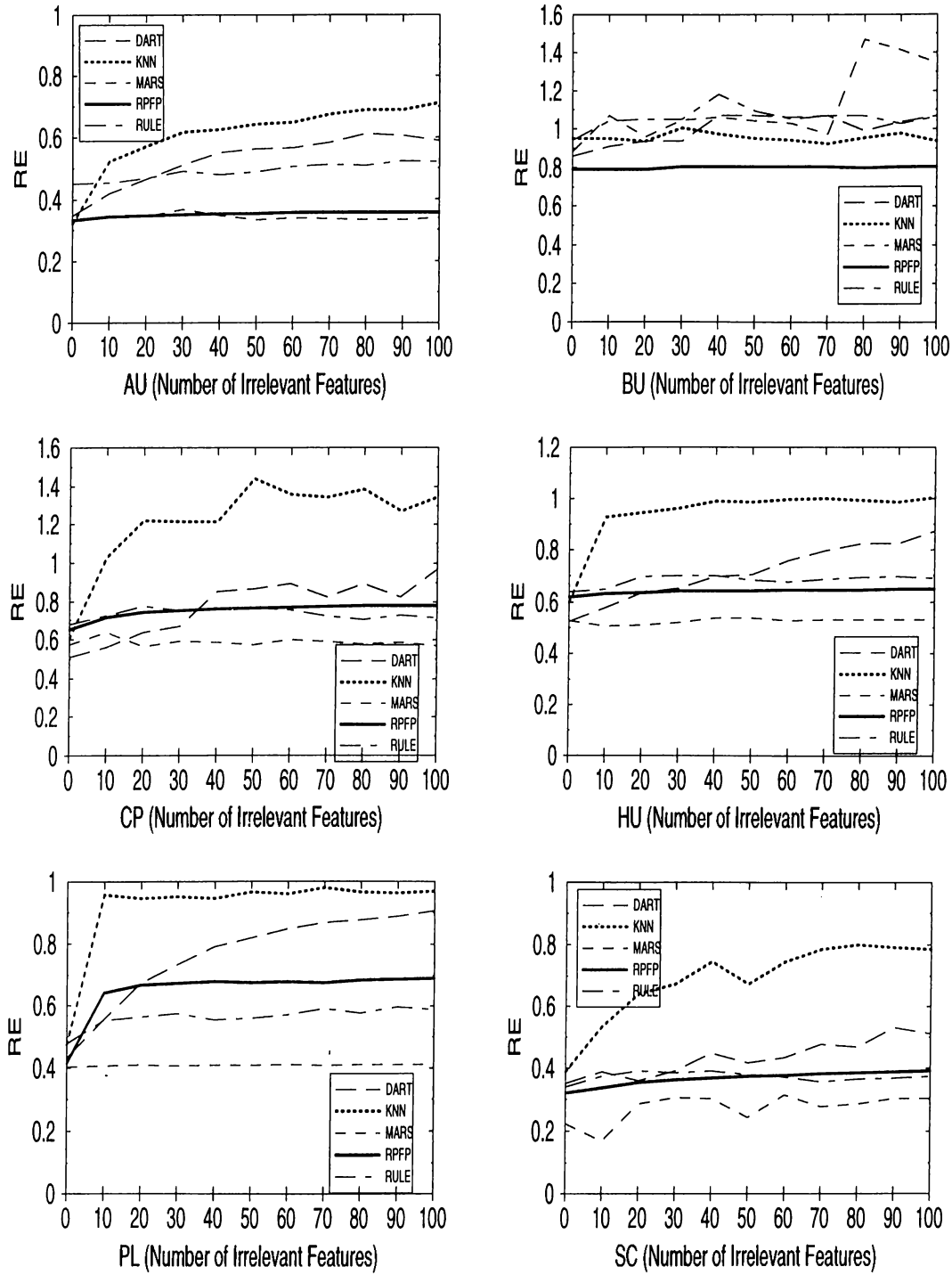


Figure 4.1. Relative errors of algorithms with increasing irrelevant features.

making computations on each feature separately.

A comparison of algorithms on all data sets where 30 irrelevant features are added to each of them is shown in Table 4.3. RFPF and MARS outperform other algorithms for the robustness to irrelevant features according to this table.

4.6 Robustness to Missing Values

With current relational databases, the issue of missing values is a common problem for most domains. RFPF handles missing values naturally by simply ignoring them, and using all other values available. A comparison of RFPF with other algorithms for increasing missing values on selected data sets is shown in Figure 4.2. As the values are removed from the data, information loss and performance degradation become obvious. However, the decrease in performance is smaller in RFPF than other algorithms, where the missing values are filled with means or most frequent nominal value. The error rate of RFPF becomes relatively minimal in all selected data sets, when proportion of missing values reaches 90%, except for low dimensional PL data set. According to these results, DART also performs well in robustness to missing values.

A comparison of algorithms on all data sets, where 20% of the values of real data sets are removed, is shown in Table 4.4. According to these results RFPF outperforms other algorithms in terms of robustness to missing values.

4.7 Robustness to Noise

It is apparent from the graphs in Figure 4.3 that RFPF-N outperforms other algorithms for most of the selected data sets. An interesting result is that RFPF also achieves better than other algorithms in most data sets. However, all algorithms except RFPF-N reaches unacceptable error rates with a small increase in the ratio of noise.

Data Set	RFPF	KNN	RULE	DART	MARS
AB	0.704	0.906	0.899	*	0.682
AI	0.500	1.539	0.744	0.658	0.682
AU	0.351	0.618	0.451	0.511	0.369
BA	0.670	0.723	0.668	0.641	0.573
BU	0.802	1.005	0.944	0.938	1.049
CL	0.716	1.320	0.290	0.306	2.195
CO	1.330	3.027	6.307	1.662	4.126
CP	0.753	1.214	0.678	0.668	0.590
EL	1.018	1.076	1.528	1.236	1.134
FA	0.698	1.058	0.820	0.877	0.249
FI	0.295	0.985	0.258	0.350	0.208
FL	1.038	1.537	1.792	1.490	1.629
FR	0.959	1.075	1.558	1.430	1.777
GS	0.568	0.893	0.218	0.573	0.404
HO	0.876	0.974	0.890	1.165	0.847
HU	0.642	0.963	0.641	0.653	0.521
NO	1.024	1.071	1.250	1.157	1.370
NR	0.979	1.149	1.217	*	0.916
PL	0.674	0.952	0.477	0.734	0.407
PO	0.775	0.934	0.916	1.013	1.005
RE	1.033	1.060	1.352	1.311	1.042
SC	0.362	0.673	0.341	0.391	0.305
SE	0.589	1.021	0.229	0.650	0.798
ST	0.782	1.151	0.906	0.756	0.818
TE	1.617	2.455	4.195	2.709	5.614
UN	0.671	0.856	0.550	0.906	0.394
VL	0.972	1.111	1.267	1.307	1.257
Mean	0.793	1.161	1.162	0.964	1.147
Variance	0.084	0.258	1.659	0.271	1.429

Table 4.3. Relative errors of algorithms, where 30 irrelevant features are added to real data sets. If the result is not available due to singular variance/covariance matrix, it is shown with (*). Best results are typed with bold font.

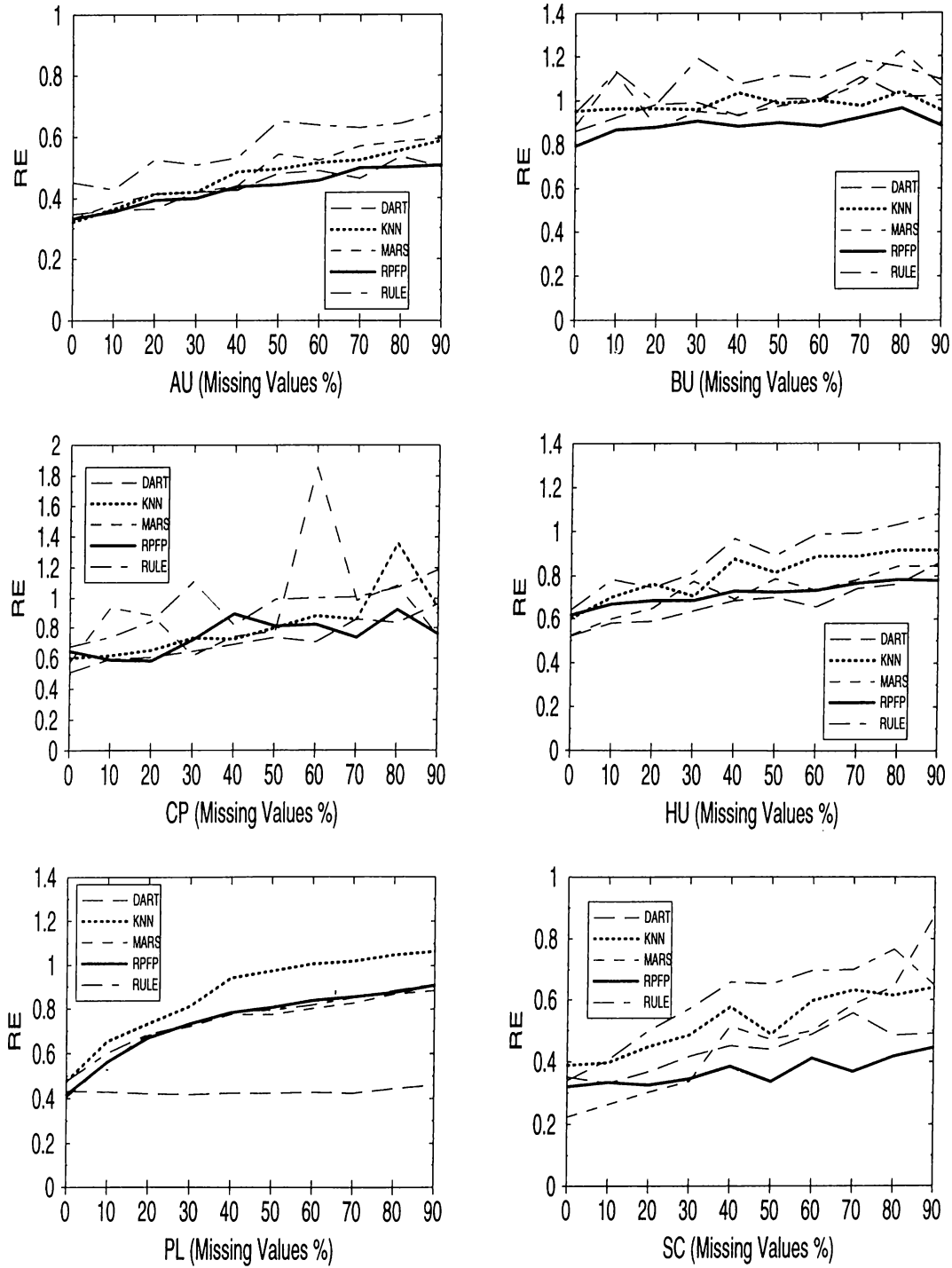


Figure 4.2. Relative errors of algorithms with increasing missing values.

Data Set	RFPF	KNN	RULE	DART	MARS
AB	0.739	0.750	0.962	0.688	0.748
AI	0.532	0.726	0.676	0.546	0.798
AU	0.393	0.414	0.526	0.363	0.414
BA	0.817	0.560	0.783	0.565	0.709
BU	0.881	0.964	0.989	0.983	0.877
CL	0.796	0.942	0.400	0.435	0.801
CO	1.439	1.856	3.698	2.377	3.733
CP	0.584	0.652	0.843	0.607	0.880
EL	1.029	1.097	1.537	1.191	1.074
FA	0.767	0.849	0.949	0.735	0.731
FI	0.273	0.584	0.336	0.320	0.348
FL	1.377	1.851	1.751	1.421	1.557
FR	1.033	1.711	1.557	1.347	1.012
GS	0.702	0.743	0.497	0.536	0.595
HO	0.889	0.911	1.040	0.974	0.836
HU	0.687	0.761	0.748	0.590	0.649
NO	0.986	1.229	1.363	1.222	0.989
NR	0.940	1.072	1.272	*	0.972
PL	0.668	0.733	0.686	0.420	0.679
PO	0.682	0.976	1.189	0.792	1.026
RE	1.007	1.059	1.364	1.229	1.048
SC	0.327	0.449	0.500	0.370	0.303
SE	0.938	0.921	0.849	0.495	0.733
ST	0.777	0.744	0.904	0.707	0.930
TE	1.810	4.398	3.645	2.512	16.503
UN	0.669	0.559	0.620	0.844	0.497
VL	1.014	1.056	1.410	*	1.090
Mean	0.843	1.058	1.152	0.891	1.501
Variance	0.110	0.587	0.670	0.323	9.372

Table 4.4. Relative errors of algorithms, where 20% of values of real data sets are removed. If the result is not available due to singular variance/covariance matrix, it is shown with (*). Best results are typed with bold font.

RFPF is generally more robust when compared to other methods even it produces complex models, that may cause large variance error which is sensitive to noise. The reason for that may be the decrease in variance error since the feature predictions are computed with different neighbors in each feature dimension. This probably makes bootstrapping effect on prediction. Bootstrapping is a common method to decrease variance error by using multiple overlapping subsets of the same data instead of a single training set.

4.8 Interactions

RFPF handles interactions in a similar way as the other eager partitioning approaches work, that is by partitioning the instance space. The best way to show how partitioning in RFPF handles interactions and generally increase accuracy for data sets having interactions is to compare it with its additive version. All other algorithms compared in the previous chapters have this property. The following experiments show that RFPF also has this property as other eager partitioning algorithms.

The additive version of RFPF is obtained by excluding the partitioning from RFPF algorithm and simply by combining the feature predictions and obtaining the final prediction after the first step. We denoted the additive version as RFPF-A.

The first experiment is done with a simple artificial data set having two interacting features and 100 instances formed as shown in Figure 4.4. Here x_1 and x_2 are the input features and y is the target. The feature x_1 takes binary values while x_2 and y take continuous values from 0 to 50. The relative error of RFPF on this data set is 0.31, which is much smaller than that of RFPF-A, whose relative error is 1.35.

Another experiment is conducted on real data sets, and the results are shown in Table 4.5. The results show that RFPF significantly outperforms RFPF-A, which indicate the ability of RFPF in handling interactions.

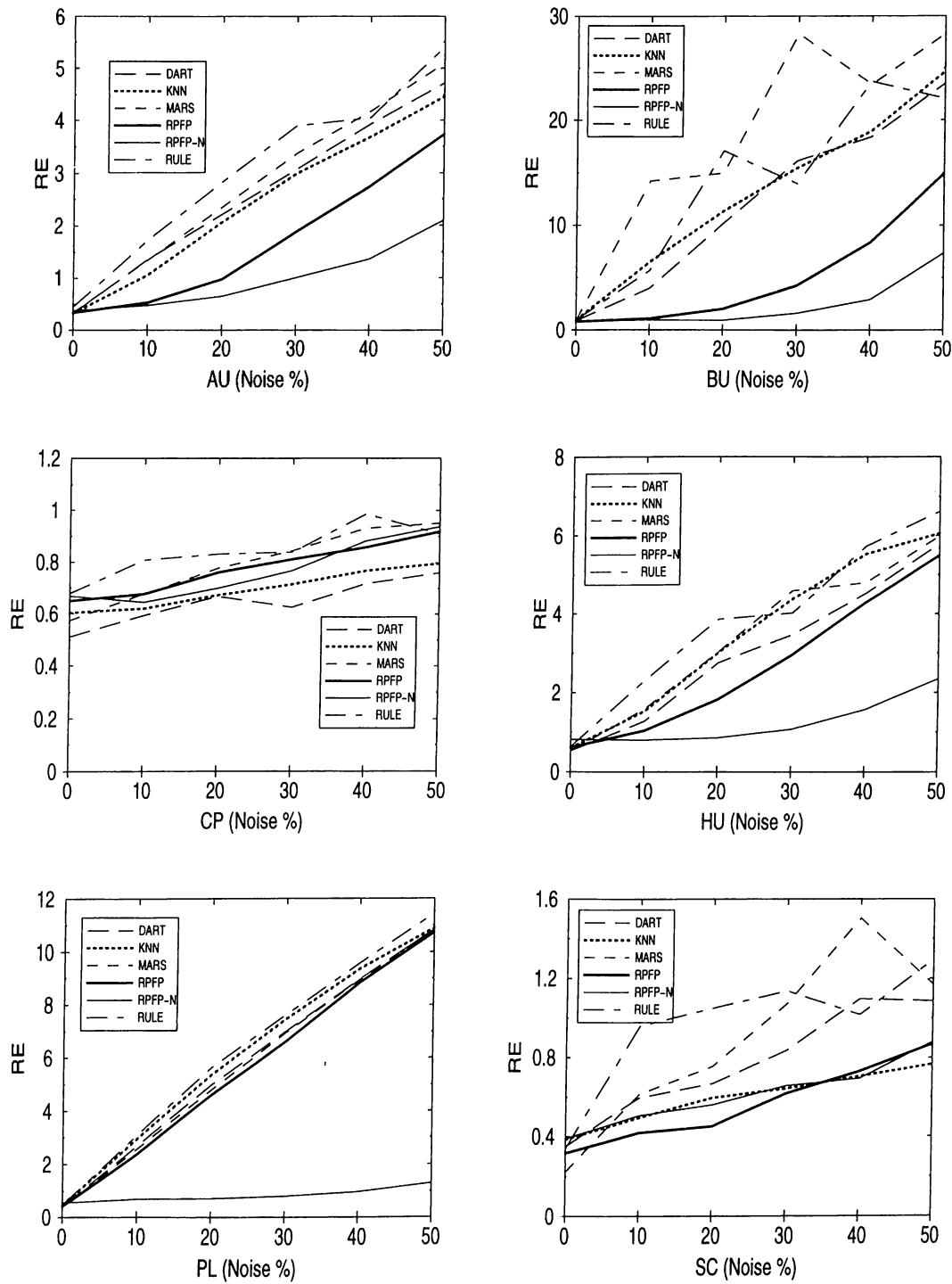
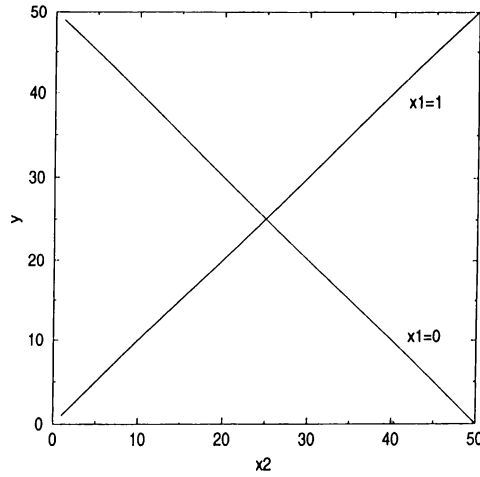


Figure 4.3. Relative errors of algorithms with increasing target noise.

Figure 4.4. Artificial data set. x_1 and x_2 are input features.

D	RFPF	RFPF-A	D	RFPF	RFPF-A	D	RFPF	RFPF-A
AB	0.675	0.815	FA	0.667	0.855	PL	0.415	0.819
AI	0.473	0.500	FI	0.243	0.334	PO	0.703	0.783
AU	0.334	0.430	FL	1.218	1.487	RE	1.008	1.000
BA	0.664	0.752	FR	1.056	1.041	SC	0.319	0.337
BU	0.792	0.896	GS	0.566	0.667	SE	0.527	0.944
CL	0.692	0.773	HO	0.868	0.939	ST	0.729	0.992
CO	1.301	1.354	HU	0.618	0.710	TE	1.659	1.629
CP	0.650	0.738	NO	0.962	0.958	UN	0.666	0.718
EL	1.009	1.019	NR	0.947	0.956	VL	0.970	0.988

Table 4.5. Comparison of RFPF with its additive version RFPF-A. Best results are typed with bold font.

4.9 Computation Times

Since the computation times of lazy and eager approaches differ significantly for training and prediction phases, training times of eager approaches and prediction or test times of lazy approaches are given in Table 4.6. Generally test times of eager approaches and training times of lazy approaches are close to zero. The time durations are measured on a Pentium450 personal computer running Linux operating system.

The results justify the theoretical considerations in determining the computational complexity of RFPF such that it is proportional to the linear prediction complexity of KNN. On the average, prediction time of RFPF is 2.5 times higher than of KNN. This is more apparent for largest datasets (AB, GS, NR). In general computation performances of algorithms differ for different datasets.

Data Set	RFPF Test	KNN Test	RULE Train	DART Train	MARS Train	RFPF/KNNR Ratio
AB	40081.2	17217.1	6593.3	458503.0	7629.1	2.3
AI	7.5	3.0	248.4	57.8	153.5	2.5
AU	124.1	41.8	407.4	1772.2	573.9	3.0
BA	261.9	50.1	429.8	3022.1	912.4	5.2
BU	51.6	49.2	284.9	667.7	738.6	1.0
CL	150.9	35.6	464.0	708.6	1039.9	4.2
CO	32.0	31.7	396.5	459.4	484.8	1.0
CP	30.6	161.0	251.4	263.6	361.0	0.2
EL	141.1	19.5	389.1	933.2	385.6	7.2
FA	167.0	30.3	403.9	1654.4	755.8	5.5
FI	18.1	161.3	278.5	200.5	226.7	0.1
FL	1198.8	775.9	408.7	901.4	543.8	1.5
FR	9.0	44.0	234.5	42.8	99.9	0.2
GS	14241.0	6435.2	1236.6	23845.8	8797.0	2.2
HO	92.8	140.7	266.3	835.8	616.2	0.7
HU	449.2	98.9	594.4	7576.7	1186.2	4.5
NO	6.0	1.7	236.8	18.0	68.1	3.5
NR	24027.0	9346.7	7006.4	81984.0	4207.3	2.6
PL	1536.4	1415.5	503.3	9343.1	670.7	1.1
PO	6.1	2.0	250.3	41.1	121.6	3.1
RE	2717.0	674.6	625.2	35541.5	2260.1	4.0
SC	7.5	2.0	251.2	78.2	283.8	3.8
SE	6.9	4.0	221.6	78.2	109.0	1.7
ST	1173.8	759.8	845.6	16203.2	1839.2	1.5
TE	0.1	0.0	235.4	3.1	25.5	*
UN	6459.7	3858.9	4834.2	153959.0	7287.0	1.7
VL	2113.9	1229.5	1101.0	107661.0	3082.9	1.7
Mean						2.5

Table 4.6. Time durations of algorithms for real data sets in milliseconds.

Chapter 5

Conclusion and Future Work

In this thesis we have presented a new regression method called RFPF. It is an instance-based, non-parametric, nonlinear, context-sensitive, and local supervised learning method based on feature projections. It achieves higher accuracy results especially when compared to the most common lazy approach, KNN. Its performance is also significant when compared to important eager approaches of both machine learning and statistics. The main drawback of RFPF is the lack of interpretation and its high prediction time requirement, as other lazy approaches.

Even though, RFPF is a lazy method, it eliminates most drawbacks of other lazy methods. The most important one is that it is robust to irrelevant features. The local weight associated with each feature enables this property. Besides, it is context-sensitive that is contribution of features are computed separately in different regions of the instance space. Some features may be important only some regions of the instance space.

RFPF also properly handles most problems belonging to all types of learners, eager or lazy. Those are curse of dimensionality, missing feature values, and noise, handled by a modification on RFPF algorithm. These are important and common problems especially with large databases today. RFPF outperforms all other important methods used in comparisons on domains having large number of missing values or noise.

The advantages and limitations of RFPF is described in the previous chapter. Future work can be directed to overcome these limitations and to incorporate new properties to RFPF. Possible improvements are described in the following paragraphs.

Redundant Features: If there are functional dependencies between some features, or some of the features give the same information, they can be factorized to a single feature, or some of them can be removed from the training set.

Interpretation: Lack of interpretation of the underlying data is a common drawback of all lazy approaches. However further research can be directed for RFPF in order to determine relative importance of features by using the local weights of features determined for each query instance. Similar work can be done in order to determine interactions between features, by using the changes in the local weights of features at each partitioning step.

Incorporating Domain Knowledge: The main motivation to develop machine learning algorithms or knowledge discovery tools is to extract knowledge without an expert, since number of domains in the databases is large. However for stand-alone applications, where the data belongs to a single domain, and where a domain knowledge is available, incorporating domain knowledge to these automatic tools, including RFPF, may increase the accuracy significantly.

Misclassification Cost: Incorporating misclassification costs to classification algorithms is a current research topic. Misdiagnosing a patient as healthy is much important fault than vice versa. Given a misclassification function for continuous target values, similar research can be directed for regression algorithms, including RFPF.

Feature Weighting and Selection: RFPF employs an implicit local weight for each feature at each step of RFPF algorithm. Incorporating feature weights computed by external weighting algorithms is not researched as well as feature selection algorithms.

Classification Tasks: Some authors describe classification as a sub-category of regression. By associating a feature having binary values for each class value of a categorical target, the performance of RFPF for classification tasks can

be evaluated.

Bootstrapping: Bootstrapping is a sampling method used to increase the performance of learning algorithms by decreasing the variance error. However for lazy approaches, where variance error is small when compared to bias error, this method does not work and they are called stable because of this property. The same thing may not occur for RFPF since it an adaptive partitioning method, and whether boosting increase its performance can be researched.

As a final word, instance-based regression by partitioning feature projections is a successful technique in regression. RFPF method can compete with the most famous and successful methods of both machine learning and statistical literature. Some important properties of RFPF that are missing in many important other methods such as handling missing values naturally, robustness, and domain independence enable it to become an important tool for knowledge discovery and data mining systems.

Bibliography

- [1] Adeli, H., Knowledge Engineering Vol.1, *McGraw Hill*, 1990.
- [2] Aha, D., Kibler, D., and Albert, M., Instance-based Learning Algorithms, *Machine Learning*, 6, 37-66, 1991.
- [3] Aha, D., Special Issue on Lazy Learning, *Artificial Intelligence Review*, Vol.11, No.1-5, 1997.
- [4] Aha D. W., Salzberg, S. L., Learning to Catch: Applying Nearest Neighbor Algorithms to Dynamic Control Tasks, *Selecting Models from Data: Artificial Intelligence and Statistics IV.*, New York, NY: Springer-Verlag, 1994.
- [5] Akkuş, A. and Güvenir, H. A., k Nearest Neighbor Classification on Feature Projections, In *Proceedings of the 13th International Conference on Machine Learning*. Lorenza Saitta (Ed.), Bari, Italy: Morgan Kaufmann, 12-19, 1996.
- [6] Akkuş. A., Batch Learning of Disjoint Feature Intervals. Bilkent University, Dept. of Computer Engineering and Information Science, MSc. Thesis, 1996.
- [7] Atkenson, G. C., Moore, A. W., Schaal, S., Locally Weighted Learning, *Artificial Intelligence Review*, Vol.11, No.1/5, 11-73, February 1997.
- [8] Blake, C., Keogh, E. and Merz, C. J., UCI Repository of Machine Learning Databases, [<http://www.ics.uci.edu/mlearn/MLRepository.html>], Irvine, CA: University of California, Department of Information and Computer Science, 1998.

- [9] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. Classification and Regression Trees. *Wadsworth*, Belmont, Calif., USA, 1984.
- [10] Breiman, L., Bagging Predictors, *Machine Learning*, 24, 123-140, 1996.
- [11] Dasarathy, B. V. (Ed.), Nearest Neighbor(NN) Norms: NN Pattern Classification Techniques, *CA:IEEE Computer Society Press*, Los Alamitos, 1991.
- [12] De Boor, C., A Practical Guide to Splines, *Springer*, New York, 1978.
- [13] Demiröz, G., Non-Incremental Classification Learning Algorithms Based-on Voting Feature Intervals, Bilkent University, Dept. of Computer Engineering and Information Science, MSc. Thesis, 1997.
- [14] Domingos, P., Occam's Two Razors, The Sharp and the Blunt, *Proceedings of KDD'98*, 1998.
- [15] Domingos, P., Context-sensitive Feature Selection for Lazy Learners, *Artificial Intelligence Review (Special Issue on Lazy Learning)*, 11, 227-253, 1997.
- [16] Duda, R. and Hart, P.E., Pattern Classification and Scene Analysis, *Wiley*, 1973.
- [17] Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., Advances in Knowledge Discovery and Data Mining, *AAAI Press*, MIT Press, Menlo Park, Cal., 1996.
- [18] Fayyad, U.M., Piatetsky-Shapiro, G., and Smyth, P., The KDD Process for Extracting Useful Knowledge from Volumes of Data, *Communications of the ACM*, Vol.39, No.11, 27-34, 1996.
- [19] Friedman, J. H., Multivariate Adaptive Regression Splines, *The Annals of Statistics*, 19 (1) 1-141, 1991.
- [20] Friedman, J. H. and Stuetzle, W., Projection Pursuit Regression. *J. Amer. Statist. Assoc.*, 76, 817-823, 1981.

- [21] Friedman, J. H. and Tukey, J. W. A Projection Pursuit Algorithm for Exploratory Data Analysis, *J. Amer. Statist. Assoc.*, C-23, 9, 881-890, 1974.
- [22] Friedman, J. H., On Bias, Variance, 0/1-loss and the Curse of Dimensionality, *Data Mining and Knowledge Discovery*, 1, 55, 1997.
- [23] Friedman, J. H., Local Learning Based on Recursive Covering, [ftp://stat.stanford.edu/pub/friedman/dart.ps.Z], 1996.
- [24] Güvenir, H. A. and Şirin, İ., Classification by Feature Partitioning, *Machine Learning*, 23:47-67, 1996.
- [25] Güvenir, H. A. and Koç, H. G., Concept Representation with Overlapping Feature Intervals, *Cybernetics and Systems: An International Journal*, 29(3), 263-282, 1998.
- [26] Güvenir, H. A. and Demiroz, G., Ilter N., Learning Differential Diagnosis of Erythemato Squamous Diseases using Voting Feature Intervals, *Artificial Intelligence in Medicine*, 13:147-165, 1998.
- [27] Güvenir, H. A., Altıngövdü S., Uysal, İ., and Erel E., Bankruptcy Prediction Using Feature Projection-based Classification, *Proceedings of Fifth International Conference on Information Systems Analysis and Synthesis*, Florida, 1999.
- [28] Güvenir, H. A. and Uysal, İ., Bilkent Function Approximation Repository, [http://funapp.cs.bilkent.edu.tr/], 1999.
- [29] Hall, P., On Projection Pursuit Regression, *The Annals of Statistics*, Vol. 17, No.2, 573-588, 1989.
- [30] Hastie, T., Tibshirani, R., Generalized Additive Models (with discussion) *Statistics Science*, 1, 297-318, 1986.
- [31] Holte, R. C., Very Simple Classification Rules Perform Well on Most Commonly Used Datasets, *Machine Learning*, 11,63-91, 1993.
- [32] Imielinski, T., Mannila, H., A Database Perspective on Knowledge Discovery, *Communications of the ACM*, Vol.39, No.11, 58-64, 1996.

- [33] Karalic, A., Employing Linear Regression in Regression Tree Leaves, *In Proceedings of ECAI'92 (European Congress on Artificial Intelligence)*, Vienna, Austria, Bernd Newmann (Ed.),440-441, 1992.
- [34] Karalic, A., Linear Regression in Regression Tree Leaves, *In Proceedings of ISSEK'92 (International School for Synthesis of Expert Knowledge) Workshop*, Bled, Slovenia, 1992.
- [35] Kaufman, L. and Rousseeuw, P.J., Finding Groups in Data - An Introduction to Cluster Analysis, *Wiley Series in Probability and Mathematical Statistics*, 1990.
- [36] Kibler, D., Aha D. W., Albert, M. K., Instance-based Prediction of Real-valued Attributes, *Comput. Intell.*, 5, 51-57, 1989.
- [37] Lock, R. H. and Arnold, T., Datasets and Stories: Introduction and Guidelines, *Journal of Statistics Education [Online]*, 1 (1), [<http://www.amstat.org/publications/jse/v1n1/datasets.html>], 1993.
- [38] Matheus, C. J., Chan, P. K., Piatetsky-Shapiro, G., Systems for Knowledge Discovery in Databases, *IEEE Transactions on Knowledge and Data Engineering*, 5(6), 903-913, 1993.
- [39] McTear, M.F., Anderson, T.J., Understanding Knowledge Engineering, *Ellis Horwood*, 1990.
- [40] Mitchell, T.M., Machine Learning, *McGraw Hill*, 1997.
- [41] Niblett, T. and Bratko, I., Learning Decision Rules in Noisy Domains, *Developments in Expert Systems*, Cambridge University Press, 1986.
- [42] Peterson, M., Lundberg, D., Ek, A., Knowledge Engineering System Design in Diffuse Domains, *Studentlitteratur, Chartwell-Bratt*, 1990.
- [43] Press, W. H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., Numerical Recipes in C, *Cambridge University Press*, 1988.
- [44] Quinlan, J. R., Combining Instance-based and Model-based Learning, *International Conference on Machine Learning* , pp. 236-243, 1993.

- [45] Quinlan, J. R., Learning with Continuous Classes, *In Proceedings AI 92 (Adams and Sterling Eds.)*, 343-348, Singapore, 1992.
- [46] Quinlan, J. R., Induction of Decision Trees, *Machine Learning*, 1, 81-106, 1986.
- [47] Quinlan, J. R., Unknown Attribute Values in Induction, In A. Segre (Ed.), *In Proceedings of the 16th International Workshop on Machine Learning*, 164-168, San Mateo, CA:Morgan Kaufmann, 1989.
- [48] Quinlan, J. R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, California, 1993.
- [49] Rawlings, J. O., Applied Regression Analysis: A Research Tool, *Wadsworth*, Belmont, California, 1988.
- [50] Rosenbrock, H.H., An Automatic Method for Finding the Greatest or Least Value of a Function, *Computer Journal*, 3, 175-184, 1960.
- [51] SPSS Sample Data Sets, [<ftp://ftp.spss.com/pub/spss/sample/datasets/>], 1999.
- [52] Tukey, J. W., Exploratory Data Analysis, *Addison-Wesley Publishing Company, Inc.*, 1977.
- [53] Uysal, İ. and Güvenir, H. A. An Overview of Regression Techniques for Knowledge Discovery, *Knowledge Engineering Review*, Cambridge University Press, Vol.14, No.4, 319-340, 1999.
- [54] Uysal, İ. and Güvenir, H. A., Regression by Feature Projections, *Proceedings of Third European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'99)*, Springer-Verlag, LNAI 1704, Jan. M. Zytkow and Jan Rauch (Eds.), Prague, Czech Republic, 568-573, 1999.
- [55] Uysal, İ. and Güvenir, H. A., An Application of Inductive Learning for Mining Basket Data, *Proceedings of TAINN'99*, İstanbul, 30-39, 1999.
- [56] Webb, G. and Kuzmycz, M. Further Experimental Evidence Against the Utility of Occam's Razor. *Journal of Artificial Intelligence Research*, 4, 397-417, 1996.

- [57] Weiss, S. and Indurkha, N., Optimized Rule Induction. *IEEE Expert*, 8(6), 61-69, 1993.
- [58] Weiss, S. and Indurkha, N., Rule-based Regression, *In Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pp1072-1078, 1993.
- [59] Weiss, S. and Indurkha, N., Rule-based Machine Learning Methods for Functional Prediction, *Journal of Artificial Intelligence Research*, 3 383-403, 1995.
- [60] Weiss, S. and Indurkha, N., Predictive Data Mining: A Practical Guide, *Morgan Kaufmann*, San Francisco, 1998.
- [61] Wettscherec, D., Aha, D.W., and Mohri, T., A Review and Empirical Evaluation of Feature-Weighting Methods for a Class of Lazy Learning Algorithms, *AI Review*, Vol.11, No.1-5, 273-314, 1997.