REGRESSION LEAF FOREST: A FAST AND ACCURATE LEARNING METHOD FOR LARGE & HIGH DIMENSIONAL DATA SETS

by

SIVANESAN GANESAN

(Under the Direction of Maria Hybinette)

ABSTRACT

There are a number of learning methods that provide solutions to classification and regression problems, including Linear Regression, Decision Trees, KNN, and SVMs. These methods work well in many applications, but they are challenged for real world problems that are noisy, non-linear or high dimensional. Furthermore, missing data (e.g., missing historical features of companies in stock data), is not managed well by current approaches. We present an implementation of a hybrid learning system that combines an ensemble of decision trees (Random Forest) with of Linear Regression. Linear Regression (LR) is fast but not accurate because it assumes linearity, while Random Forests are not as fast as LR but have been shown to be accurate for high dimensional and large data sets. By combining these approaches we address the weaknesses of each approach and exploit their strengths both in terms of real time performance and accuracy.

In this thesis, we evaluate a hybrid Random Forest and Linear Regression implementation called "Regression Leaf Forest", which is a forest of trees with regression leaves for supervised

learning problems. The approach extends Random Forests by introducing Linear Regression learners at the leaf nodes of the trees for predicting functions. Our empirical analysis on both real and artificial data shows that the proposed algorithm requires less computation time for both large and high-dimensional datasets while providing comparable or better accuracy when compared to: Single Tree, a Single Linear Regression Tree, and Random Forest algorithms.

INDEX WORDS: Random Forest, Linear Regression, Regression Leaf Forests

REGRESSION LEAF FOREST: A FAST AND ACCURATE LEARNING METHOD FOR LARGE & HIGH DIMENSIONAL DATA SETS

by

SIVANESAN GANESAN

B.E., Anna University - India, 2006

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2011

© 2011

Sivanesan Ganesan

All Rights Reserved

REGRESSION LEAF FOREST: A FAST AND ACCURATE LEARNING METHOD FOR LARGE & HIGH DIMENSIONAL DATA SETS

by

SIVANESAN GANESAN

Major Professor:	Maria Hybinette
Committee:	Eileen T. Kraemer
	Shelby Funk

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2011

DEDICATION

This work is dedicated to my **mother, father** and **advisor** who have supported me all the time. Without their moral support, it would not be possible for me to be in this position. Also, my dedication goes to beloved **GOD**, whom I believe gave me the strength, courage and will power throughout my life. Finally I dedicate it to my **friends** and **well-wishers** who were always there for my needs.

ACKNOWLEDGEMENTS

Foremost, I would like to express my deep and sincere gratitude to my advisor **Dr. Maria Hybinette** for her continuous support and encouragement. Dr. Hybinette has always been very kind and caring for her students. Her sage thoughts, insightful criticism and patient encouragement aided me in completion of this thesis. Throughout my thesis-writing period, she provided excellent support, sound advice, good company, and lots of good ideas. I would have been lost without her.

Besides my advisor, I would also like to heartily thank my committee members **Dr. Eileen Kraemer** and **Dr. Shelby Funk** for their time and consideration. I would like to extend my special thanks to **Dr. Tucker Balch** for his helpful suggestions and consultations throughout this research.

Last but not the least I would like to thank my family for supporting me spiritually throughout my life.

TABLE OF CONTENTS

Page

APPENDICES

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

1.1 Background

Machine Learning (ML) algorithms are of great practical value in a variety of domains, including: (1) data mining domains of large databases that contain regularities that can be discovered automatically (e.g., to analyze outcomes of medical treatments from patient databases (Canlas, 2009)); or to learn general rules for credit worthiness from financial databases (Langdell, 2002); (2) poorly understood domains where humans might not have the knowledge needed to directly develop effective algorithms (e.g., human face recognition from images (Kamber, 2000)); and (3) domains where the program must dynamically adapt to changing conditions (e.g., controlling manufacturing processes under changing supply stocks or adapting to the changing reading interests of individuals (Mitchell T. , 1997)). Typically an ML problem requires a well-specified task, performance metric, and source of training experience.

We define learning to include any computer program that improves its performance at some task through experience. More precisely a computer program is said to learn from experience 'E' with respect to some class of tasks 'T' and performance measure 'P', if its performance at tasks in 'T', as measured by 'P', improves with experience 'E'.

Some disciplines that have influenced the development of ML include: information theory, control theory and statistics. Of these, control theory and statistics are the most applicable here. Adaptive control theory provides systems that learn to control processes in

order to optimize predefined objectives. They work by learning to predict the next state of the process they are controlling given a particular control input. Statistics provides methods for characterizing errors (bias and variance) with respect to the accuracy of a hypothesis based on a limited sample of data.

1.2 Supervised Learning & Ensemble Learners

In this work we focus on *supervised learning* in which the learning system is trained to approximate a function (e.g., Y = f(X)). The task is to predict or approximate the value of f(X). The system is provided many example input and output pairs (X, Y). After training is complete, the learner can be used to predict the value Y given an X.

Supervised learning systems can be applied to *classification* or *regression* problems. In *classification* problems the output, Y, is a category or nominal value. For instance, a learner might be trained to recognize animals in images. Relevant outputs in this case could be "zebra," "giraffe," or "horse." For regression problems the output is a real-valued number. Regression outputs might represent a temperature prediction, or a stock price.

There are a number of algorithms designed to address supervised learning problems. The most popular methods are linear regression, KNN, decision trees, and support vector machines (SVM). In most of these approaches one "model" is created based on training data, then that model is used at runtime for classification or regression prediction. This work focuses on decision trees, and ensembles of decision trees.

Decision trees have a number of features that make them well suited to a number of ML tasks:

- Ability to handle high dimensional (large number of attributes) data

- Natural handling of data of “mixed” type (enumerate and real)
- Robustness (noise and erroneous data)
- Computational scalability (large N)
- Ability to deal with noisy, irregular or 'missing' data

1.3 Random Forests

Recent research has focused on how to improve supervised learning by combining a group of learners that work together as a “committee.” Ensemble learners have been shown to have more power at generalization and are less susceptible to over fitting (Wolpert, 1999). The term “ensemble” can be used to describe the paradigm that brings together a number of learning machines to provide a single output.

One issue faced by those building ensemble learners is the method by which the component learners are constructed. In this work we focus on the Random Forest approach in which the ensemble is composed of multiple decision trees (thus a “forest”). Each tree is composed using a randomized technique to ensure they are independent (Breiman L. , 2001).

Random forests consist of many decision trees. At run time, in order to find an output, each tree in the ensemble is queried using the same X input (where X is a vector of features). In the case of a classification problem the outputs of the trees can be combined by “voting” where the classification with the most votes “wins,” e.g., for a forest with 10 trees if “zebra” gets 7 votes and “giraffe” gets 3, “zebra” wins. For regression problems, the outputs are combined in other ways, such as by mean, mode or median.

Decision trees (and forests) sometimes suffer from low prediction accuracy and over fitting. This limitation is overcome using ensemble methods (forests) in a number of ways. One

approach is to build a large set of simpler trees where the leaves represent more of the data. Normally the data at the leaves is summarized by averaging the values in some way – but in this work we introduce a new approach where the leaves are learners themselves. In particular, we replace the leaves with linear regression learners.

1.4 Regression Leaf Forest

The approach followed in this research contributes to the combination of two different techniques: Random Forest and Linear Regression. The trees have linear regression learners at the leaves. Our intuition is that Random Forests perform well with large multi-dimensional data sets and can also address the challenge of finding discrete as well as continuous functions, but it's comparatively slow whereas Linear Regression's main advantage is its processing speed. So by combining these two approaches we bring a new approach called "Regression Forest Leaves" which effectively and at the same time without compromising much processing time, provides better prediction results with small classification error rates.

1.5 Problem Domain

Our research aims to solve the problem of predicting multi-dimensional large data sets with *noise and missing values for continuous functions*. When it comes to data with multiple dimensions, many systems suffer from the "curse of dimensionality", where the performance degrades as the dimensions increase. Similarly, it is often found to have less accurate predictions with increase in noise and missing values or attributes, which affects real world problems such as medical data and financial applications. A goal in our research is to address these issues without compromising the system performance.

1.6 Contributions

Our contributions in this research include:

- A new scalable, fast and accurate hybrid algorithm that combines the advantages random forests and linear regression
- A new method dealing both missing and noisy data accurately (low error)
- A comprehensive empirical study that includes large-scale data of multiple dimensions both with real world and synthetic data.

The rest of the thesis is organized as follows: Chapter 2 describes in detail our proposed approach and all the related information. Chapter 3 discusses related work done by various other researchers in this area. The experiments are discussed and the results are presented in Chapter 4. Finally, Chapter 5 presents the conclusions of this thesis and describes future work.

CHAPTER 2

APPROACH: REGRESSION LEAF FOREST AND RATIONALE

In this chapter we describe our hierarchical approach called Regression Leaf Forests (RLF) and its rationale. Regression Leaf Forest is a hybrid approach that combines random forests (Breiman and Cutler's) algorithm (Breiman & Cutler, 2001) with linear regression learners at the leaf nodes. To provide insight into our approach and our implementation we will first discuss random forests (and decision trees) then discuss linear regression, and conclude with a discussion of our approach. Regression Leaf Forest is an ensemble of forest of trees with regression leaf nodes.

In this chapter we focus on details relating to our approach, for other details, e. g., on related work concerning tree growing techniques, or finding the best node at each split, or details concerning the over-fitting problem please refer to the related work section of this thesis. In our examples below we consider both classification (discrete classes) and continuous data (regressions, where the predictions are defined by either value or a defined by a function).

2.1 Decision Trees

As previously discussed a decision tree enables making informed decisions (or predictions, or classifications) from a set of observations, because it provides a mapping from a set of observations X of tuples, where an X_i in X is of the form: $<X_1, X_2 \ldots , X_M>$, and X = { X_i

in X| $<X_1, X_2 ... , X_M>$, $<X_1, X_2 ... , X_M>$, ... $<X_1, X_2 ... , X_M>$}, to a conclusion or a outcome Y, such as Y = f(X). A tuple X is a set of features that influences the outcome Y.

As an example, consider the task of classification of an EKG. The numerical readings are X, and here, a decision tree could be used to automatically identify a Y; In this case whether a person has a normal result or one of several types of arrhythmia depending on the measurements of X (Canlas, 2009). The Xs are the measurements and Y is the predicted outcome. The tree is parsed from top (root) to bottom (leaves), with branches going left or right according to measurements.

The decision tree is 'trained' using data that has been labeled by an expert. A labeled training example is a group of measurements (e.g., a person's EKG results) paired with a classification (e.g., "arrhythmia type A"). As an example, in the EKG application mentioned above, we might record the assessments of recognized experts of EKGs from several hundred patients, and then use their assessments as labels for training.

After training with labeled examples, the decision tree may be queried for automatic decision-making or classification of arrhythmia. Assume that a decision tree for this data exists. We will now outline the procedure for consulting it. The general process of making a prediction is to consider a set of 'features' of the data (this is X) and then traverse a prebuilt decision tree as follows:

1. Start at the root node of the tree.
2. Follow one branch or the other depending on the result of a single question or decision at that node (e.g., is $X_1 < 0.5$?)
3. Arrive at a new node

4. If the new node is a leaf, we are done; the value is stored at the leaf node, if not we go back to step 2 and repeat.

As another illustrative example, consider a wine company that wants to evaluate the quality of their wine to determine how much they can reasonably charge for their wines (a visualization of the three building process for our example is depicted in Figure 1). Suppose that the wine company has measured a number of properties, X_1, X_2, X_3, ... , X_M of their wine, including: alcohol content, hue, sugars, citric acid, and color intensity. Furthermore they have the result of wine taste experts that rank wines from 0 (worst) to 3 (best). Here the observed inputs are the properties of the wines, which are defined by Xs as tuples, and the outputs, Ys which are the expert's quality ranking of the wine. Suppose the wine *training* set consists of 100 instances (each instance is a tuple) of wines with measured properties and their resulting expert ranking. The data set consists of more than 100 instances but we use 100 wines for training.

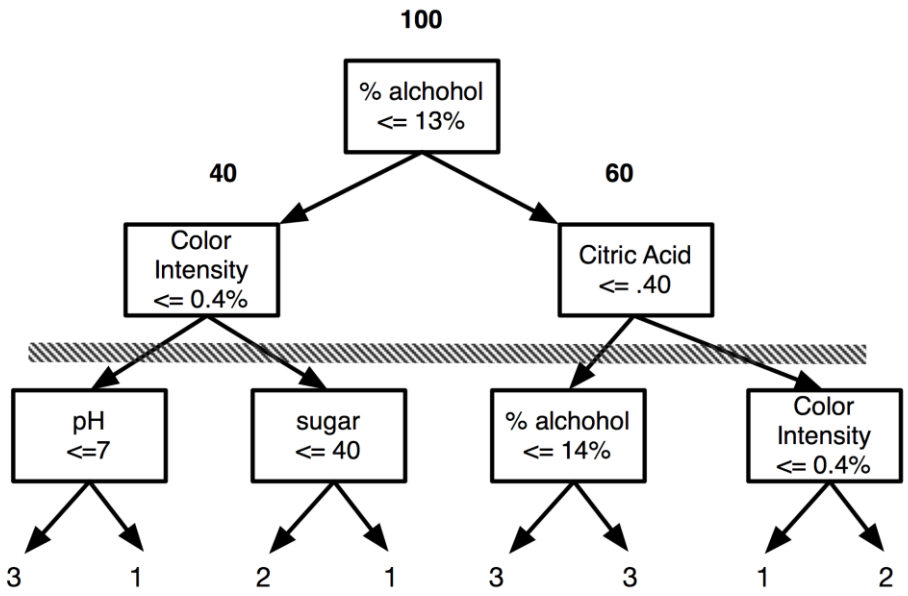

Figure 1: Decision tree for wine classification.

2.2 Building the Decision Tree

To build a tree from the training data one starts with a full subset of data (in our example the subset is if 100 instances of wine) and then recursively splits the data into subsets of data (bottles of wine) until each subset has one remaining data member, and these form the leaf nodes in the tree.

For our wine examples suppose that the first split (at the root) is on alcohol content so that wines with less than 13% alcohol are filtered to the left and the remainder of the wines are filtered to the right. Using the training data, suppose that 40 wines are filtered to the left (because they have 13% or less alcohol content) and 60 wines are filtered to the right. Before continuing, we will discuss on what value one should split on - we used 13% in our example. There are really two questions that we need to consider: (1) Value Selection: "How is that value selected?" and (2) Variable or Feature Selection: "What property should we select to split on at each branch?" We will first consider which value to split on first; next we will consider variable selection.

2.3 Values and Feature Selection:

Each node represents a binary question: Is *feature* K less than or equal to split *value* Z? There are two general ways to select a *value* to split on, one is to use the median of all remaining bottles in the set so that approximately half the wines are filtered in each direction, and another approach is to randomly select one, two or more wines, and then take the average (or some other statistic - e.g., the median) of their X value (feature K, alcohol content) and use that value as the split value (Quinlan J. R., 1986). In our approach the split value is in accordance to the second method; take the mean of value of the specific attribute of a number of randomly selected tuples.

There are at least three general approaches for *feature* selection at a node: (1) is to randomly select a feature, (2) is to select of subset (m, where m<<M) of features, then among the subset compute the feature that is most strongly correlated to Y, and (3) is to examine all features (M) at a node and the split on the feature that is most strongly correlated to Y. The subset for (2) and (3) can be determined by random sampling, i.e., repeated sampling until the desired number of data is in the subset.

The idea behind (2) and (3) is to pick a 'representative' attribute to split on, this is determined by first picking a random subset by a repeated sample of the remaining data set at the node and then computing the "most important" attribute from the random subset. Typically this is the attribute that correlates most to the quality of the wines remaining in the branch. Note that between the simpler approaches and the approaches that use correlation that there is a time trade-off - as the more complex approach requires more computation. An advantage of randomly selecting on features to split nodes is robustness to noise and missing features.

To continue with our example we consider the next iteration, now we split each of the two smaller two sets filtered left and right of the tree. Suppose we work on the left branch and suppose that there are now 40 wines on the left on 60 wines on the right (See Figure #). Now we again randomly select a feature to split the wines into another two smaller sets (e.g., we may split on color intensity). Suppose that our randomizer selects to split on color intensity with a value of 50% then we split the wines that have less than 50% intensity to the left and the larger values to the right. The process is continued until each branch split set contains only one single wine, this set of one forms a leaf in the decision tree (in the figure X the tree is truncated to a depth of 3 for simplification). The outcome at the leaf is then the quality ranking of the remaining wine labeled by the expert.

In this example a leaf is of size one, there may be cases however, when recursion on the training sample should stop before reaching a data size of 1, e.g., when (1) the depth of the tree has reached a specified maximum value, (2) the number of training data tuples remaining in a node is less than a specified threshold, or when the number of tuples is not statistically representative to split the node further, or (3) all the remaining tuples in the node belong to the same class (or, in the case of regression, the variation is too small).

2.4 Training Data

The tree building process is called the training phase. An issue that we have not yet addressed concerns how to pick a 'good' training set or subset of data from a larger set. What is a good training set? Goals in picking a good training set include (1) Enable accurate prediction of the remainder of the data set (not used for training) and (2) Enable accurate prediction of data that is *not in the data set*. An intuition to enable accurate prediction of data outside the available data is to randomize the training set to avoid bias towards the available data. To reduce bias we adopt *bootstrapping* (outlined below and in Figure 2) proposed by Breiman and Cutler (Breiman & Cutler, 2001) in 2001. Here an initial training set (a subset) is first selected from the initial data set by repeatedly selecting tuples randomly (here a tuple can only be picked once, so it is without replacement), then a second training set is generated from this initial training set by randomly sampling from the data set (here a tuple can be selected multiple times, so it is by replacement), the second set is the same size as the first set. The second generated training set is sometimes called the *bootstrap set*, and the training process itself is called *bootstrapping.*

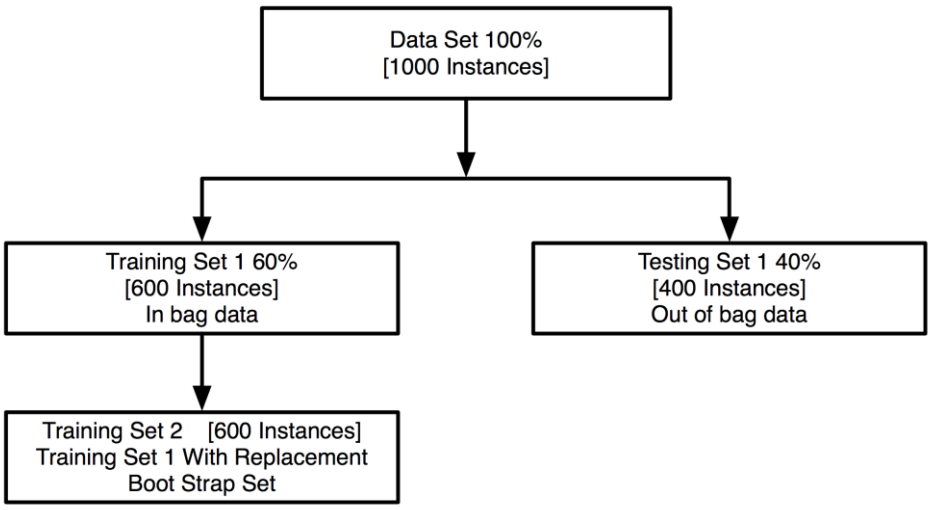

Figure 2: Flow diagram showing bootstrapping during training.

Creating a two-phase generation of the training set and separating the training data from the testing set ensure an unbiased estimate of the classification or regression (if we are evaluating continuous data). The testing set is sometimes called *out-of-bag* data to indicate that it is separate from the data used in the training phase (Breiman & Cutler, 2001).

2.5 Testing the Decision Tree

After the tree is built we can query the tree for a prediction on the quality of the wine and also find out how well the algorithm performs by evaluating its accuracy. The data set not used for training is used for testing. Suppose one picks a 'test' wine (not in the training set), then traverses the tree according to the features at each branch to predict the quality of the wine - it is assumed that this test wine has not been ranked by an expert (yet), but of course in reality it has a quality value, this value is later used to measure the accuracy of the prediction. In our example, suppose we picked a wine including the following features: 12% alcohol, a color intensity of

.6%, and sugars of value 20. If we use the decision tree depicted in Figure X, this wine should have a predictive value or grade of 2, a middle range wine. The outcome is determined by traversing the tree by first going to the left (alcohol of 12% is less than 13%), then traversing to the right (color intensity of .6% is greater than .4%), and finally going to the left (sugars of 20 is less than 40), then reading the leaf value which is 2 (a midrange wine). So here we predict that the test wine receives the same value of the wine in the decision tree (that corresponds to the remaining single wine used in the training phase).

In practice, decision tree algorithms are evaluated by comparing the value or quality of the wine selected from the testing phase with the quality computed using the decision tree. Typically, mean square error is a common measure to evaluate how well an algorithm predicts the quality of a test sample, but other error methods may be applied depending on the metrics being evaluated.

2.6 Decision Tree Forests

A decision forest is an ensemble learner that combines the results of a number of trees to provide higher quality result (lower error) and to avoid over fitting. Over fitting is when the model predicts the behavior of noise instead of the characteristics of the data. In the case of a regression (continues values instead of classes) problem, a number of decision trees are parsed or queried then the results of the decision trees are averaged. For example, in the wine example we use the same test wine and traverse each tree in the forest (or a subset of trees in the forest). The value of the final leaf node of each tree is then averaged to generate the predicted quality of the wine.

To train a forest we use 60% (which is common practice) of the data as a training set, then for each tree, we pick 60% from the training set (so we again take 60% of a set) through sampling the original training set, next we generate the 'bootstrap' set from the training set that samples the set with replacements. As a consequence this set can include duplicates as previously discussed.

In our hierarchical approach, once we have grown a specified number of trees as explained according the above procedure of decision trees; we say that we have a fully-grown forest.

Our approach in building a forest is summarized in Figure **3,** and the brief description is outline below. Our first step is to discretize continuous data. We use user-defined bins, which is common see e.g., (Shinomoto 2007) or (Scott 1979), as a vehicle to discretize. Next we create a training set by repeated sampling of data until 60% of the data make up the training set, the remaining data is the testing set, also called the out of bag data. After generating the training set we build the trees in the forest. Each tree are build using a 30% sample of the training set - where each tree's sample is generated by sampling the original training set but with replacement (i.e., this training set can have repeated tuples). We then build the trees according to the process described above until we reach the desired number of trees. The number of trees in a forest is typically 10 to 100 trees, independent of the size of the data set.

Here are the steps (which are illustrated in Figure 3):

Step 1: Create training set **T** of size **N** from data set **D**, so that **N** = 0.6 x size(D). Sample the data set and randomly select N independent tuples.

Step 2: For each tree create a bootstrap training set **B** of size **N** from **T**. Sample **T**, **N** times, but with replacement, i.e., the same tuple can be picked multiple times because of replacement. This final training set is the bootstrap for the tree.

Step 3: Generate decision trees so that specific splitting attributes and values are selected as follows. The specific attribute at split is determined by evaluating **m** samples (random subset) from the bootstrap set, where m is much less than the M (number of attributes) in the original data set. Then selecting the best split amongst the m attributes.

Step 4: Each tree is grown to the fullest extent, i.e., until one cannot split the data set further, after a tree is grow repeat step 2 to grow another tree.

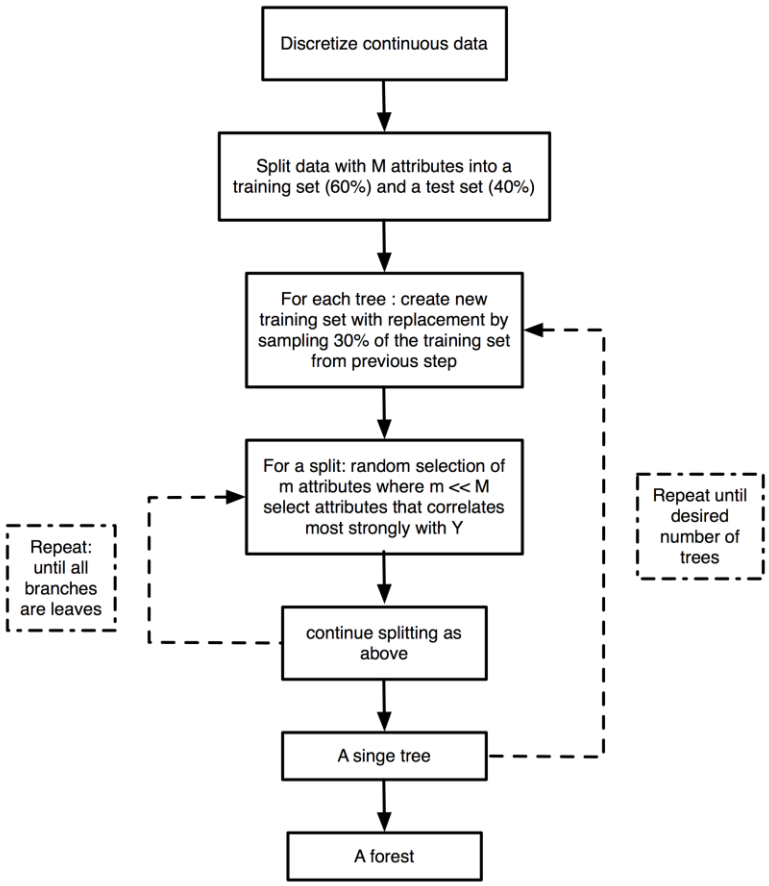

Figure 3: Flow chart on building a forest of trees.

2.7 Testing the Decision Tree Forest

After the generation of the tree, the remaining data set (the tuples not used for training) is used to test the individual trees as well as the entire forest. The average 'misclassification' or error is called the out-of-bag error estimate. This error estimate is useful for predicting the performance of the machine learner. Some random forest algorithms use a weighted forest, where trees carry different weights and 'heavier' trees have a higher impact than trees with lesser weight. In our approach each tree are given equal weight.

2.8 Decision Tree Forests: Advantages and Disadvantages

We will later examine quantitatively some of the advantages and disadvantages of decision forests. Here is a quick preview: Advantages of decision trees include: (1) they are easy to implement (2) they are resistant to noise and missing data and (3) they are fast. However, these advantages are weighted against disadvantages. Some disadvantages of random forests are:

- Decision tree forests are more complex and cannot be as easily visualized as a single tree.
- Even though random forests can handle high dimensional data, it is computationally slow when compared with linear models.
- Required significant memory for computation compared to simpler models (e.g., linear regression models).

Linear regression (LR) can address some of these disadvantages, and we discuss LR below.

2.9 Linear Regression

Linear regression is simple and fast. Our approach is to combine the advantages of linear regression with the advantages of random forests (noise and missing data resistance). In this section we provide an introduction to LR starting from one dimension and then building from there. Linear regression enables us to determine a relationship between a continuous process output (Y) and its attributes (X_1, X_2, ... X_M). To start simple let's consider a one-dimensional data set X= {X_1} (note that number of dimensions is analogous to the number of attributes), the relationship can be expressed as:

$$Y = b + m X_1$$

In this equation Y is the outcome, or the predicted value of the dependent variable X_1, m is the slope of the regression line, b is the Y-intercept of the regression line. The general idea behind linear regression is to find a line that most closely models the data (see Figure 4 that plots data and the regression line). This can be done by least squares that minimize the sum of squared differences between observed values (the *y* values) and predicted values. We use a library in python that computes LR using least squares.

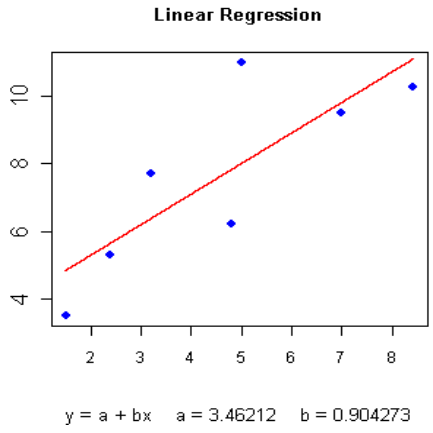

Figure 4: Linear Regression

A linear regression for multidimensional data is expressed as follows:

$$Y = b_0 + b_1 X_1 + b_2 X_2 + ... + b_k X_k + e$$

Here, Y is predicted value and $X_1, X_2, ... , X_k$ are the attributes and e is error. $b_0, b_1, b_2, ... , b_k$ are known as the regression coefficients, which are estimated from the data. In our approach we use multi-dimensional LR at the leaves of decision trees.

2.10 Regression Leaf Forests

In our approach we combine linear regression with random forests, here the general idea is that decision trees have learners (linear regression learners) at the leaves instead of single values as in a random forest. A way to visualize a Regression Leaf Forest (RLF) is that the leaves in RLF fits piecewise interpolating surfaces or blocks to the predicted function, while the linear regression refines fitting these blocks so that they flush against the predicted function for a better fit (or it conforms more closely to the underlying function). Ensemble learning has the potential to offer at least two advantages: a reduction in over fitting (more accurate) and it is fast and robust because of simplicity, while linear regression can improve accuracy and further improve performance.

Earlier approaches such as (Karalic, 1992), (Dobra and Gehrke, 2002), and (Vogel et al., 2007) have shown that regression trees with learning models at the leaves (of single trees) can improve both on accuracy and speed, however these approaches are still prone to over fitting and do not scale well in practice. It should be noted that these studies are based on learners of single trees or classifiers and are not ensembles that are less prone to over fitting. There are no studies that we are aware of that apply learners at leaves of a forest.

2.11 Regression Leaf Forests Building

The steps of building a regression leaf forest follows a similar process as building trees in a random forest except that computation terminates at a node when there are d_p instances left in the training set, where d_p is a parameterized variable that determines number of remaining instances instead of the standard single instance. The remaining (1+) instances form a leaf cluster. Figure 5 illustrates this process, the boxes at the leaves of the trees denote leaf clusters that are of size d_p or less. Different learners at leaves are attached to the clusters; in this work we primarily apply linear regression to the leaf clusters.

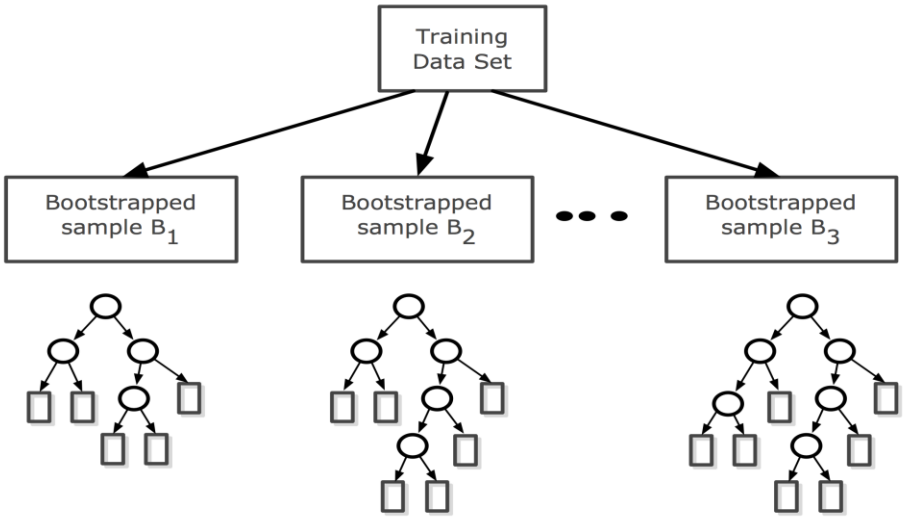

Figure 5: Building a regression leaf forest.

2.12 Testing the Regression Leaf Forests

At the testing phase, the forest is queried. When reaching a "leaf cluster" the instances are fed as input to a linear regression algorithm, which computes a predicted value Y for the particular tree. Of course we can also make the process simpler and instead of applying a linear regression to the cluster we could apply the average (or the median) of the remaining instances in

the cluster (e.g., if computation time is of essence). Here the trade-off is accuracy and time. Continuing from a decision tree to the trees in the forest, we get the output values (from the linear regressions) for all the trees in the forest and then average their result to get our target predicted (Y). The testing algorithm of our approach is listed below:

Step 1: Query each tree and get a cluster C_i of instances.

Step 2: Apply linear regression to each C_i and get a predicted value Y_i.

Step 3: Average the values of each Y_i from each tree in the forest and output Y_p with error statistics

2.13 Challenging Data Sets

A challenge for classification algorithms are that training data may contain missing attribute values (NaNs) or noisy data. Linear regression typically can handle noisy data well, but is challenged both with missing attributes and when the data is non-linear. Fortunately Random Forrest handles both missing data (Qi, et al., 2005) and noisy data problem fairly well (Breiman, et al. 1984). We will give a description of the data set and the results in the experimental chapter.

2.14 Models with Missing Data

In our approach we address missing data as we build the trees in the forest, other approaches may preprocess their data and fill in the holes with an average value of an attribute (e.g., SVM's uses mean and KNN uses distance measure). In order to describe our approach we will give an insight on our implementation and data structures. In our example we will denote missing data with a NaN, or NaNs.

Tree nodes in RLF are stored in matrix with two dimensional arrays, the columns are the features and the rows are the corresponding values. In our wine data for example, each bottle of wine would be a row and the features of the wine would be in the columns, e.g., one column may be alcohol content. Suppose there are bottles (tuples) that lack alcohol content. While building the tree we need to address two problems: (1) How is data split when we pick a sample at random and its value is missing, i.e., it is NaN? And (2) how do we deal with the situation when an entire training data set lacks a feature (i.e., an entire column is filled with NaN or is missing)?

To answer the above questions we use a new approach called 'LrNaNed' model to handle missing data (or NaNs). Our model transforms data that contains missing values (NaNs) to the new data, free of NaNs. The LrNaNed keeps track of all the data with its original values and the newly replaced corresponding values in case if it's a NaN.

In practice LrNaNed is a wrapper to a conventional linear regression model. It is specifically designed to handle NaNs because linear regression cannot by default handle missing data. Once the missing data is handled and replaced by LrNaNed it is passed to the linear regression method to find the best fit with minimal error.

2.15 Wrapper Class LrNaNed()

To give more detail of the wrapper class, the implementation of LrNaNed primes the data at the leaves before performing a linear regression. It replaces the NaNs with the median value of the feature values in the training sets. So the primary step for LrNaNed is to find the median 'm_i' of each attribute.

The procedure is to get their medians and store them in arrays. Since arrays can be accessed in constant time if the index is known, it gives us good performance for retrieving the

median of any required data value using the index. In the case of an entire column missing (containing NaNs), we simply replace it with 0's and perform the linear regression as before.

In summary during the training phase, at each leaf the linear regression creates a set of coefficients and at the same we calculate the median of each attribute and store them for future use (LrNaNed). So during testing phase, we query the learned model and if our query has NaN in it, we will simply replace the NaN with its appropriate median (which was already computed in the previous step) and then calculate the new response using the coefficients that we computed during the training phase to get our predicted target value $Y_{p.}$

CHAPTER 3

RELATED WORK

This section surveys and reviews various research works on learning algorithms including SVM, KNN, Neural Networks, Linear Regression, and Random Forest as well as how different types of algorithms have been useful in the field of data prediction and classification. There has been a recent trend towards learning algorithms particularly with random forest because of its accuracy and robustness. Here we present some of the related work done by researchers in this area. We will survey the work in classification and regression and also work that address noisy and missing data.

3.1. Classification and Regression

There are many effective learning methods available for predicting values including CART (Breiman, Friedman, Olshen, & Stone, 1984) that builds regression trees that differ from decision trees only in having values rather than classes at the leaves. MARS (Friedman, 1988) similarly constructs models whose basis functions are splines. The M5 (Quinlan J. R., 1992) trees introduced by Quinlan in 1992, can handle multi-dimensional data, which makes it more powerful than CART and MARS, as they suffer from the curse of dimensionality. The M5 for learning models predict values like CART, where M5 builds tree-based models where the trees can have multi-variate linear models at branches, which are analogous to piecewise linear functions while simple regression trees will have values only at their leaves. While M5 trees are

smaller they have shown to be more accurate compared with CART and the MARS approach. A concern with M5 trees is that regression trees cannot predict values outside the range observed in the training cases, whereas tree-based models can extrapolate.

Karlic introduced another model of adding linear regression to the regression tree leaves (Karalic, 1992). Here the idea is to grow a regression tree with leaves that implements a function $Y = f(X_1,X_2 \ldots X_n)$ of n continuous or discrete attributes. The tree growing algorithm splits an example set representing the node of tree into two subsets from which they recursively form sub trees and also used the estimate of variance of the class values as the node splitting criteria. And further, the leaves which are extended to allow linear function of continuous attributes where then introduced with linear regression. This approach of introducing local linear regression to the leaves of regression tree showed beneficial results with better accuracy by doing repetitive tests (involving many cases). However the technique showed decreased performance when analyzed with individual cases.

The SECRET (Dobra & Gehrke, 2002) mechanism proposed by Dohra and Gehrke, showed that trees with linear regression could improve on scalability and accuracy for large data sets compared to another state-of-art regression tree algorithm called GUIDE (Loh, 2002) by employing a number of techniques. The main idea is, for every intermediate node, to find two Gaussian clusters in the regressor-output space and then to classify the data points based on the closeness to these clusters. The mechanism also borrows from classification tree algorithms that locally select the split variable and the splitting point to improve performance and avoid solving a large number of linear systems requiring an exhaustive search algorithm.

The "model trees" as a general form of linear regression trees was discussed by (Vogel et al. 2007) in Scalable Look Ahead Linear Regression Trees (LLRT) mechanism. Here the models

at each leaf node of the partitioned data set may be predictive model; not necessarily a linear regression model. LLRT brings in a scalable approach to exhaustively evaluate all possible models in the leaf nodes in order to obtain an optimal split. LLRT performs a near-exhaustive evaluation of the set of possible splits in node: splits are evaluated with respect to the quality of the linear regression models. The high increase in performance of LLRT for handling practically large data sets is that it efficiently calculates the k-fold cross validation of large data by pre-computing the data-dependent part prior to iterating over the possible splits in current data. In spite, LLRT's minor drawback is that it is slower for small samples by a modest factor than conventional methods that we discussed.

3.2. Missing Data

Part of the motivation of our work is to address data with missing values. In this section we will review related work that addresses missing data. Recently, (Ishwaran, Kogalur, Blackstone, & Lauer, 2008) introduced Random Survival Forests (RFS) to handle missing data. The random survival forest is an ensemble tree method for analysis of right-censored survival data. They main focus on survival data is to provide high prediction performance. The algorithm strictly adheres to the principle laid out by Breiman (2003) for Random Forest. The splitting criterion used in growing decision tree must explicitly involve survival time and censoring information. Hence it is very similar to the conventional RF algorithm. Random Survival Forest handles missing data using an approach called the adaptive tree imputation. The idea here is to adaptively impute missing data as a tree is grown. Imputation works by drawing randomly from the set of non-missing in-bag data within the working node. Because only in-bag data are used for imputing, out-of-bag (OOB) data are not touched, and the OOB prediction error

is not optimistically biased. Similar way is used to handle the missing data for outcomes. RSF also iterates the process repeatedly to improve the prediction accuracy when the data has large missing values. Thus RSF illustrated high prediction accuracy with real data to uncover highly complex interrelationships between variable and showed optimal results by keeping low with root mean squared error (RMSE) and out-of-bag error for survival data with varying percentage of missing values.

Node Harvest (NH) (Meinshausen, 2009) is another approach suitable for classification and regression with multi-variate predictor variables for achieving high predictive accuracy and interpretability. An initial set of a few thousand nodes is generated randomly. If a new observation falls into just a single node, its prediction is the mean response of all training observation within this node, identical to a tree-like prediction. A new observation falls typically into several nodes and its prediction is then the weighted average of the mean responses across all these nodes. The only role of node harvest is to `pick' the right nodes from the initial large ensemble of nodes by choosing node weights, which amounts in the proposed algorithm to a quadratic programming problem with linear inequality constraints. The solution is sparse in the sense that only very few nodes are selected with a non-zero weight. Node harvest can handle mixed data and missing values and is shown to be simple to interpret and competitive in predictive accuracy on a variety of datasets. NH deals missing data with or without using imputation techniques or surrogate splits when predicting the response for new observations with missing values. Finally, regularization is proposed that can reduce the number of selected nodes. So once a fit is obtained, predictions for new data can be obtained without use of imputation techniques or surrogate splits. NH showed optimal results with low signal-to-noise data where as it showed poor fit with high signal-to-noise ratio when random forest consistently performed

better. But in summary NH showed simplicity and better interpretability of results when compared to random forest with many other data sets and more over NH was able to deal with missing values without explicit use of imputation or surrogate splits.

3.3. Other Machine Learning Techniques

Neural Networks in the field of Machine learning and Data Mining has a wide usage in time series forecasting, mostly these are feed-forward networks (Multi-Layer Perceptron) which uses a sliding window protocol over the input sequence. Typical examples of this approach are market predictions, meteorological and network traffic forecasting. Neural networks essentially comprise three pieces: the architecture or model, the learning algorithm and the activation functions (Fausett, 1994). In real-world systems the nature of the state space is obscure, hence the actual variables that contribute to the state vector are mostly unknown. Initially the window size estimation heuristics were used to determine the embedding size and time lag. The greatest strength of neural networks is their ability to accurately predict outcomes of complex problems. In accuracy tests against other approaches, neural networks are always able to score very high (Berson, Smith, & Thearling, 1999). Even though, there are some downfalls to neural networks. First, they have been criticized as being useful for prediction, but not always in understanding a model. Secondly, neural networks are susceptible to over-training. If a network with a large capacity for learning is trained using too few data examples to support that capacity, the network first sets about learning the general trends of the data. Another issue with neural networks is training speed. Neural networks require many passes to build. This means that creating the most accurate models can be very time consuming (Fayyad, Grinstein, & Wierse, 2002). In noisy time series prediction, neural networks typically take a delay embedding of previous inputs which is

then mapped into a prediction. However, high noise, high non-stationary time series prediction is fundamentally difficult for these models. In order to form a more accurate model, it is desirable to use as large a training set as possible. However, for the case of highly non-stationary data, increasing the size of the training set results in more data with statistics that are less relevant to the task at hand being used in the creation of the model. The high noise and small datasets make the models prone to over-fitting. Random correlations between the inputs and outputs can present great difficulty. The curse of dimensionality refers to the exponential growth of hyper volume as a function of dimensionality. However, it is hard to obtain dense samples in high dimensions.

The KNN (K-nearest neighbor) approach is a learning methodology in pattern recognition for classifying objects on the closest training examples in the feature space. In this model, an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its 'K' nearest neighbors. The same technique applies can be used for regression, by simply assuming the property value for the object to be the average of the values of its k-nearest neighbors. Also it is beneficial to weight the contributions to the neighbors so that the nearest points contribute more to the weighted average than the distant ones. The problem of missing data in time series is formulated as a semi-supervised learning problem. Then, the co-training algorithm which was basically designed for classification and only recently for regression, is modeled to be applied for the time series prediction problem. It's been suggested to use different base learners and their ensemble along with different confidence measures. Current experiments involve extending it to multiple predictors, various data sets and multiple base predictors. KNN is robust to noisy training data (especially if we use inverse square of weighted distance as the "distance") and effective if the training data is large, but at the same time lacks in certain aspects. KNN needs to determine value of parameter K (number of

nearest neighbors). Computation cost is quite high because we need to compute distance of each query instance to all training samples.

The support vector machine (SVM) algorithm is one of the most popular learning techniques with strong empirical performance on a wide range of classification problems. In another terms, it is a classification and regression prediction tool that uses machine learning theory to maximize predictive accuracy while automatically avoiding over fitting to the data. Furthermore, several recently described extensions allow the SVM to assign relative weights to various datasets, depending upon their utilities in performing a given classification task. The model produced by Support Vector Regression (SVR) depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction (within a threshold ε). There is also a least squares version of support vector machine (SVM) called least squares support vector machine (LS-SVM). The new model separates time series into linear part and nonlinear part, and predicts the nonlinear part by SVR. Experiments show that the new regression advances the efficiency of the forecasting compared to the common SVR. However, it's just a common case to use first-order linear model as stable part. In the future work, based on the difference characteristic, the stable part can be second-order linear or third-order linear model, so much as not to be linear, such as exponential curve model. The major strengths of SVM are the training is relatively easy. No local optimal required, unlike in neural networks. It scales relatively well to high dimensional data and the trade- off between classifier complexity and error can be controlled explicitly. The weakness includes the need for a good kernel function.

Linear regression is a statistical procedure for predicting the value of a dependent variable from an independent variable when the relationship between the variables can be

described with a linear model. Linear regression is performed either to predict the response variable based on the predictor variables, or to study the relationship between the response variable and predictor variables. While linear regression implements a statistical model, it shows optimal results when relationships between the independent variables and the dependent variable are almost linear. The problem occurs when linear regression is often inappropriately used to model non-linear relationships. Also linear regression is limited to predicting only numeric output (cannot differentiate classes or ranks). Hence with linear regression there is always a lack of explanation about what has been learned.

Piecewise Linear Regression (Ferrari & Muselli, 2002) reconstructs both continuous and non-continuous real valued mappings starting from a finite set of possibly noisy samples. The proposed learning technique combines local estimation, clustering in weight space, multicategory classification and linear regression in order to achieve the desired result. This technique proved to show low generalization Root Mean Square (RMS) error by 10-fold cross validation estimate. The Piecewise Linear Regression also shows low error estimate when compared with a trained two layer neural network. The advantage in PLR is that each segment is a two dimensional linear snapshot and can be easily plotted and interpreted. But PLR requires that explanatory factors are measured with error, which results in least squares estimators being biased and inconsistent which is a drawback.

There are several traditional methods in the prediction problems including Neural Networks, SVM and Statistical methods. But they have their limitations. Statistical method only fits simple time series, but this method isn't nice for complicated time series. Neural Networks has good learning capability, but it often causes under fitting and over fitting that the

generalization capability becomes weaker. SVM promises to be much better but has the weakness to necessitate it with a working kernel function for its accuracy.

So from our discussion so far it is evident that most of the conventional approach to the learning (classification or regression) problem suffers from the drawbacks related to either performance, accuracy, memory space complexity or the implementation. Our proposed Random Forest combined with linear regression (Regression Leaf Forest) based approach, as it will be described in detail later in this thesis, is a novel approach to the traditional learning problems with missing or noisy data.

CHAPTER 4

EXPERIMENTS & RESULTS

In this chapter we present empirical results of our Regression Leaf Forest (RLF) approach and compare its performance to other current approaches (linear regression, random forest, and (single) regression trees). Our main objective is to evaluate both prediction accuracy and real time performance. We consider both real world data and synthetic data sets. To make the discussion more clear when discussing our empirical results we will first present results of 2 and 3 dimensional, non-noisy and complete data sets and then show results of varying number of dimensions, for both rich and sparse data, different levels of noise and incomplete data. Before presenting on results we will discuss the data sets.

4.1 Data Sets

For the experimental study we used six data sets, four are real world and two are synthetic datasets. The real world data has been explored extensively in the literature, e.g., (Quinlan J. R., 1993), (Tao, 2004), (Brown, 2004) and (Cortez, Cerdeira, Almeida, Matos, & Reis, 2009), and the synthetic data sets are sinusoidal data similar to data sets that have been studied elsewhere, e.g., (Dobra & Gehrke, 2002). The datasets are described below:

Boston Housing Data Set originates from the Carnegie Mellon University and now resides at the University of California at Irvin's Machine Learning (UCI) repository.

It contains attributes (e.g., crime rate, and student to teacher ratio at schools) that affect housing values in suburbs of Boston. Consists of 13 continuous attributes, and contains 506 instances with no missing values. [http://archive.ics.uci.edu/ml/datasets/Housing].

Wine Quality Data Set is from the University of California at Irvin's Machine Learning (UCI) repository.

It contains the chemical analysis of white and red variants of wines from the Portuguese "Vino Verde" wine. Consists of 12 continuous attributes and contains 4898 instances with no missing values. This larger set is separately defined into two datasets for white and red wines. We used this set primarily for proof of concept. [http://archive.ics.uci.edu/ml/datasets/Wine+Quality].

Auto MPG Data Set originates from the Carnegie Mellon University and now resides at the University of California Irvin's Machine Learning (UCI) repository.

It contains data that describes city-cycle fuel consumption in miles per gallon. Consists of 9 continuous attributes, 6 missing values and it contains 398 instances. [http://archive.ics.uci.edu/ml/datasets/Auto+MPG].

Parkinsons Telemonitoring Data Set is from the University of Oxford and now resides at the University of California at Irvin's Machine Learning (UCI) repository.

This dataset is composed of a range of biomedical voice measurements from 42 people with early-stage Parkinson's disease. Consists of 26 continuous attributes, no missing values and it contains 5875 instances.

[http://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring]

Sinusoidal 3D Synthetic Sin Set is our first synthetic data set.

It contains 2 continuous predictor attributes uniformly distributed in the interval [−3, 3], with the output defined as Y = 3sin(X1)sin(X2). This is the same synthetic set evaluating a SECRET Single Linear Regression Tree reported in (Dobra & Gehrke, 2002). We evaluate this set both with and without noise and missing data.

Sinusoidal 3D Synthetic Cos SQRT Set is our second synthetic data set.

It contains X continuous predictor attributes uniformly distributed in interval [−1, 1], with the output defined as Y = cos(4*sqrt(x1**2+x2**2). We evaluate this set both with and without noise and missing data.

The experiments were performed on a 2.3 GHz PC with 4 Giga bytes main memory, dual core processors running under Microsoft Windows Vista Home Premium. We implemented all algorithms in python. For details on implementation of our algorithms see Appendix A. We will now present results from experiments using two and three-dimensional runs to make the results more clear in the more complex scenarios.

4.2 2D: Experiments with Two Attributes

We now illustrate by plotting a simple 2D synthetic functions in order to demonstrate the characteristics of the results and discuss how we generate the synthetic data. We will demonstrate with and without noise and also include results with missing data and compare our

results to other approaches. The table below shows the input parameters for these experiments. For the 2D case we used the sinusoidal function: Y = sin (8x).

Input	Random Forest	Random Forest Leaves
Number in Leaf Clusters	1	10
% Noise	5	5
Number of Trees	3	3
Data set (train)	100 (10%)	100 (10%)
Data set (test)	1000	1000

Table 1: Set up parameters for the 2D and 3D functions.

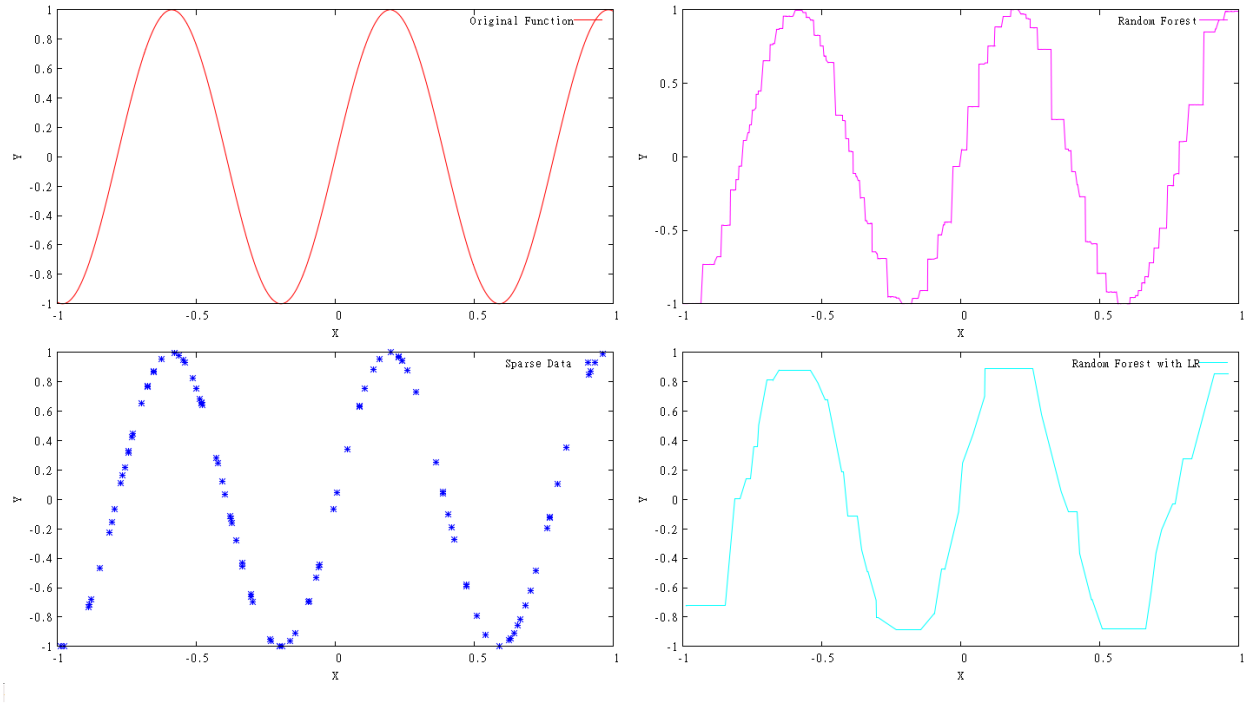

Figure 6: sin (8x) function, plotted from left to right 1. Original function 2. Sparse data 3. Random forest 4. Regression Leaf Forest

The results of the various iterations of our algorithms with the underlying sinusoidal function are shown in Figure 6. We first considered the sinusoidal function: sin(8x)) with no noise or missing values, this is shown in the top left in Figure 6. We then remove data points iteratively using a uniform distribution to create a sparser set; this is shown the bottom left of Figure 6. We use this same set to evaluate different algorithms with respect to accuracy and predictive error, the result of Random Forest is show in the upper right and the result of our algorithm is shown in the bottom right of Figure 6. Note that the Linear Forest Regression approach has a smother curve that flushed against the underlying sinusoidal function. This is

because LR approximates the value of the function between intervals better than a simple, single value as in Random Forest implementations.

We next introduce noise in the data set. We do this similarly to creating a sparse set, i.e., iteratively create noise at random instances and then introducing noise by generating a number between [-1, 1] using a uniform distribution. We plot the results of the original function with noise; the sparse function with noise in Figure 7 upper left and bottom left respectively. Note that here we created only moderate noise (i.e., only 5% noise) but the plots look quite 'noisy'. Next we use this generated sparse set with noise as input to Random Forest and to Regression Leaf Forest, the result is shown in the upper figure to the left and bottom figure to the left respectively. Note again that the plot of Regression Leaf Forest is smoother and traces the original function more closely.

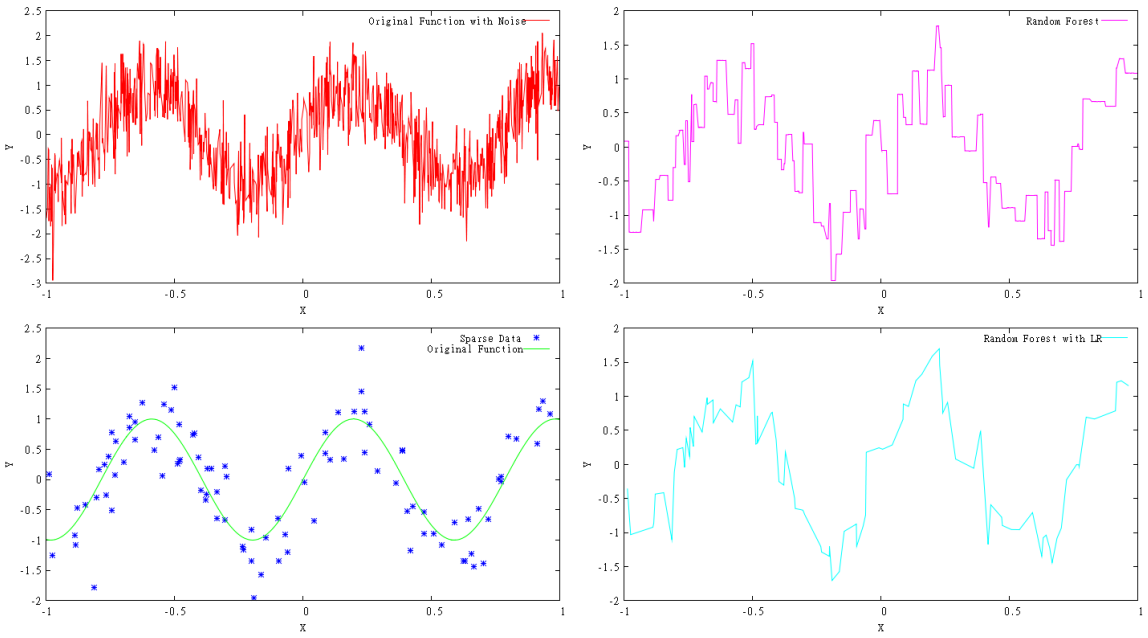

Figure 7: sin (8x) function, plotted from left to right 1. Original function with Noise 2. Sparse data 3. Random forest 4. Regression Leaf Forest.

4.3 3D: Experiments with Three Attributes

Similar to the 2D approach we now demonsrate the process by plotting a 3D synthetic functions, again to demonstrate the characteristics of the results and give details on how we generated the synthetic data. As in the 2D plots we show ou results with and without noise and also include results with missing data and compare our results to other approaches. The table below shows the input parameters for these experiments. For the 3D case we used another sinusoidal function: Y = cos (4sqrt(x1^2+x2^2). The plots X below shows the result without noise for the original function Y (upper left), then a sparse representation of the function (lower left) then the results of Random Forest (upper right) and Regression Leaf Forest (lower right).

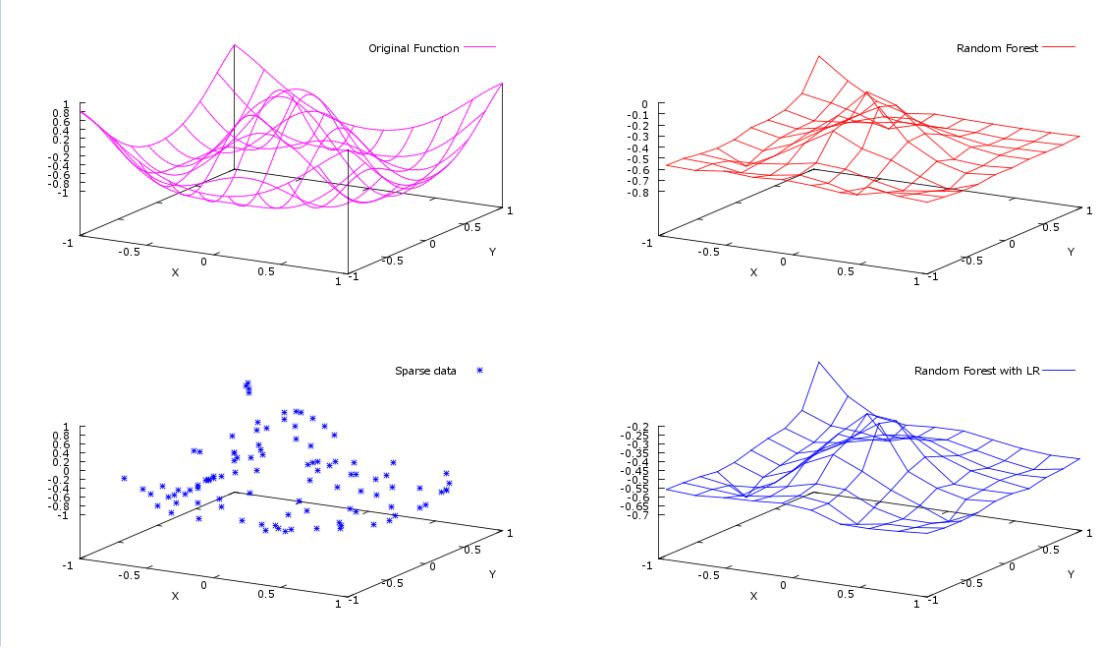

Figure 8: cos (4sqrt(x1^2+x2^2) function, plotted from left to right 1. Original function 2. Sparse data 3. Random Forest 4. Regression Leaf Forest

Note again that not surprisingly the Regression Leaf Forest plot is smoother than the plot of the simpler random forest plot. Our final plots shows the behavior of the function after introducing noise to the data sets, noise is generated similar to method in 2D plots, i.e., selecting data points uniformly then generating noise to perturb the results over the interval [-1, 1] using a uniform distribution. As in the 2D plots these plots illustrates that the original function is more discernable in the Regression Leaf Forest approach rather than the results of Random Forest.

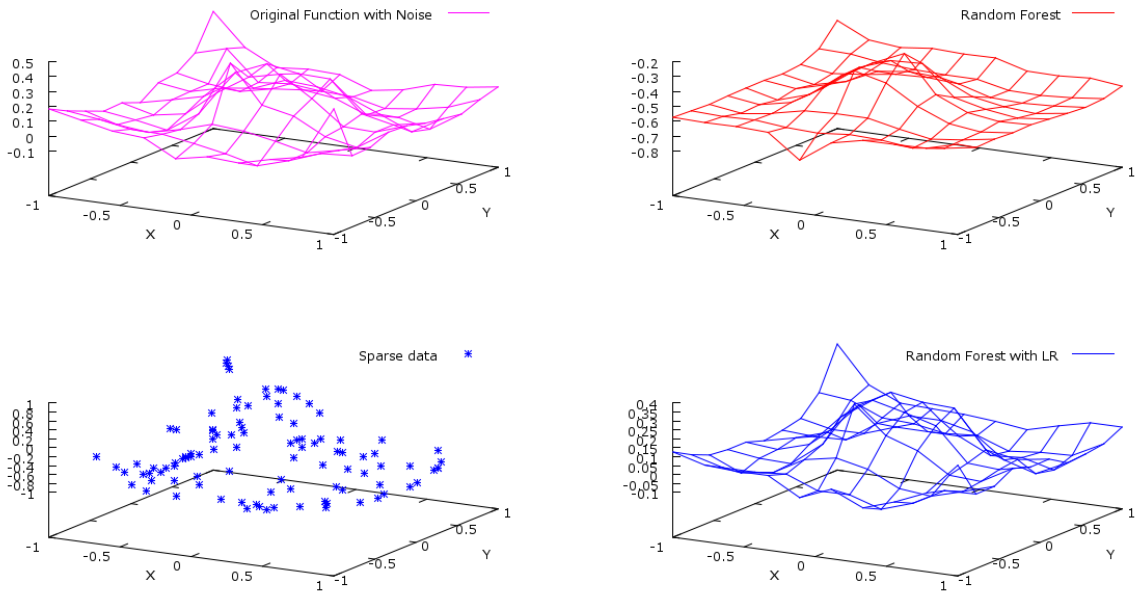

Figure 9: cos (4sqrt(x1^2+x2^2) + Noise function, plotted from left to right 1. Original function 2. Sparse data 3. Random Forest 4. Regression Leaf Forest

4.4 20D: Experiments with multi-dimensional data

In this part we evaluate the performance of our approach using of multi-dimensional data and compare it to other approaches. We evaluate both accuracy and run time (including preprocessing time). For this part we used a synthetically generated twenty dimensional set, where Y is a continuous function $f(X_1+X_2+\ldots+X_n)$, n=20. The noise level was 0, and the missing data level was 0. Each data point is the average of 5 runs.

The first set shows the result while varying the size of the data set from 100 instances up to 200,000 instances with 10 trees in the forest. Figure 10 shows the results.

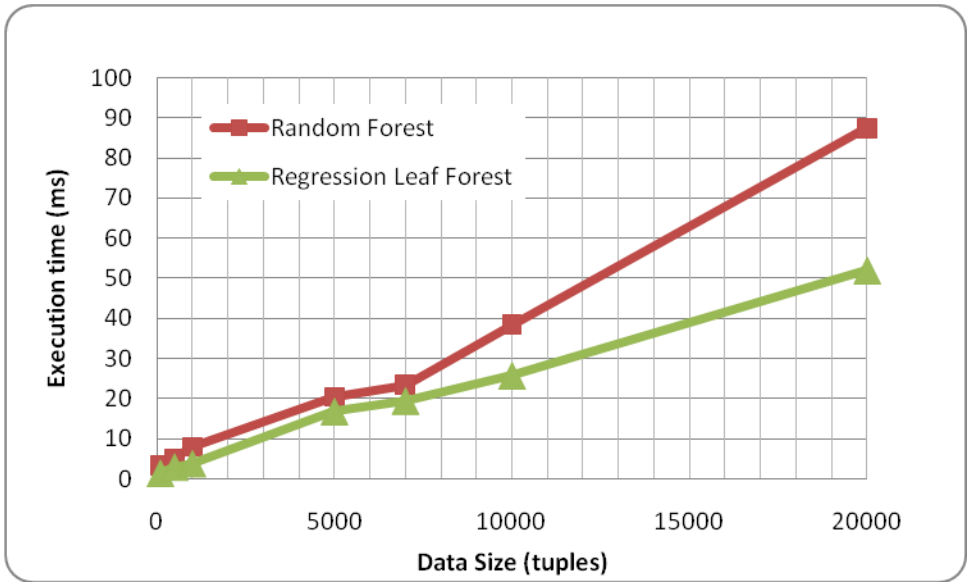

Figure 10: Performance of 20 dimensional data of Regression Leaf Forest (lower plot) and Random Forest (upper plot) as dataset increases when number of trees is 10.

Here, both Random Forest and Regression Leaf Forest perform similar up to 7,000 instances of data, as the data set increases beyond 7,000 instances of data Regression Leaf Forest outperforms Random Forest.

If number of trees in the forest increases, the performance difference increases, where Regression Leaf Forest outperforms Random Forest. The below set shows the result while varying the size of the data set from 100 instances up to 50,000 instances with 50 trees (instead of 10 as in the previous test) in the forest. Figure 11 shows the results.

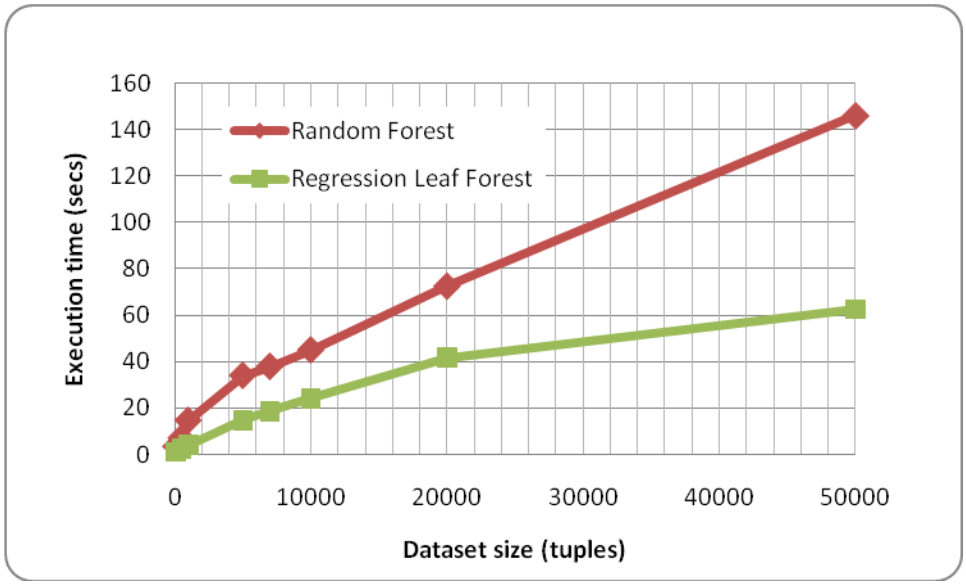

Figure 11: Performance of 20 dimensional data of Regression Leaf Forest (lower plot) and Random Forest (upper plot) as dataset increases when number of trees is 50.

The below set shows the result while varying the size of the data set from 100 instances up to 50,000 instances with 100 trees in the forest. Figure 12 shows the results, and again the performance advantage of Regression Leaf Forest increases.

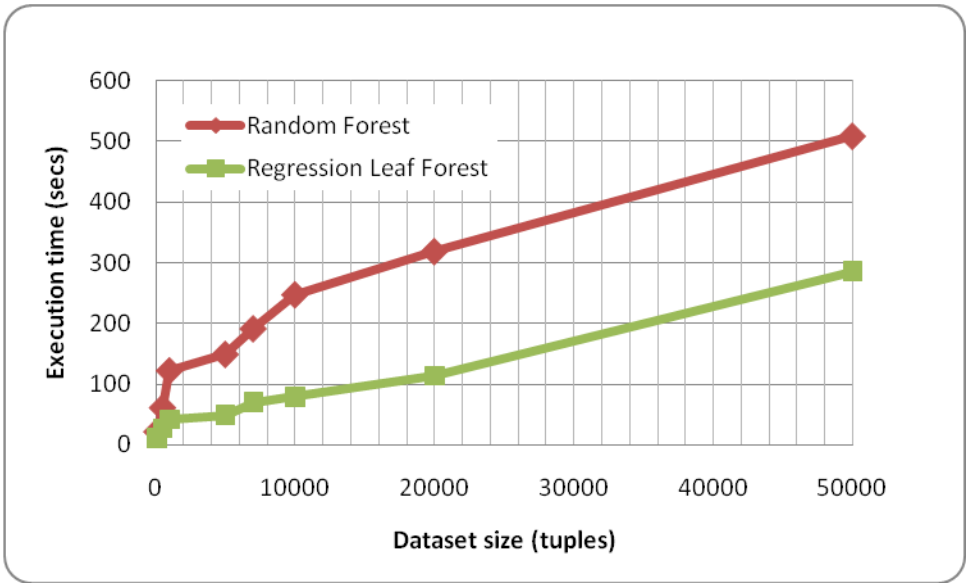

Figure 12: Performance of 20 dimensional data of Regression Leaf Forest (lower plot) and Random Forest (upper plot) as dataset increases when number of trees is 100.

For these sets of experiment Regression Leaf Forest always outperforms Random Forest and that the performance advantage increasingly improves as the data set increases (the plot diverges) and number of trees in the forest increases.

4.5 Experiments with various dimensions

In this part we evaluate the performance of our approach by varying the number of dimensions, keeping fixed data size and compare it to other approaches. We evaluate both accuracy and run time (including preprocessing time). For this part we used a synthetically generated dataset with 1,000 tuples. The noise level was 0, and the missing data level was 0. The dimensions were varied from 2 to 40. Each data point is the average of 5 runs.

The first set shows the result while varying the dimensions from 2 up to 40 with 10 trees in the forest. Figure 13 shows the described results.

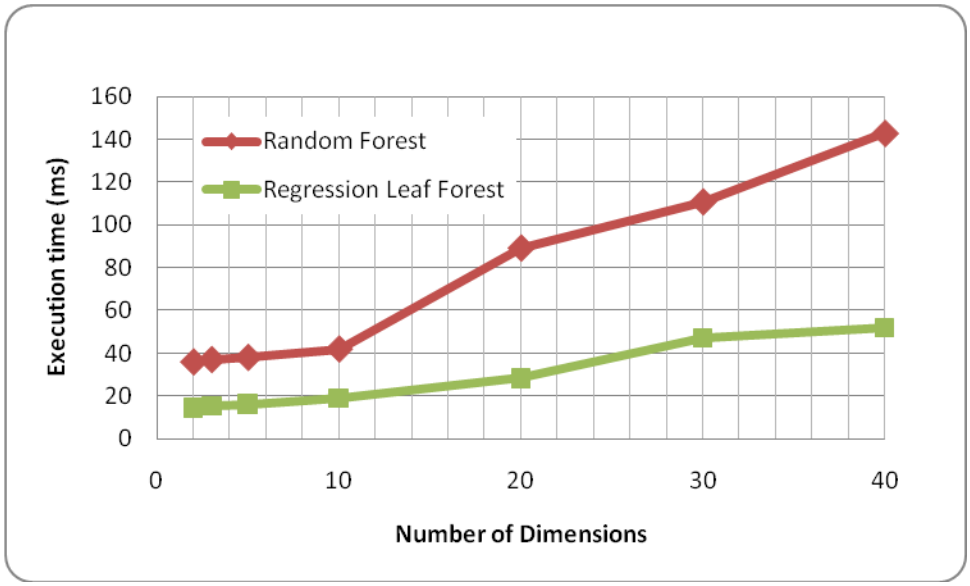

Figure 13: Performance of Regression Leaf Forest (lower plot) and Random Forest (upper plot) as number of dimensions increases when number of trees is 10.

The description of the experiment below shows the result of varying the number of dimensions (2 – 40) and keeping a fixed number of tuples (50,000). This was tested with 50 trees in the forest. Figure 14 shows the results. Here again we observe that Regression Forest Leaves outperforms Random forest and the performance advantage increases as number of dimensions increase.

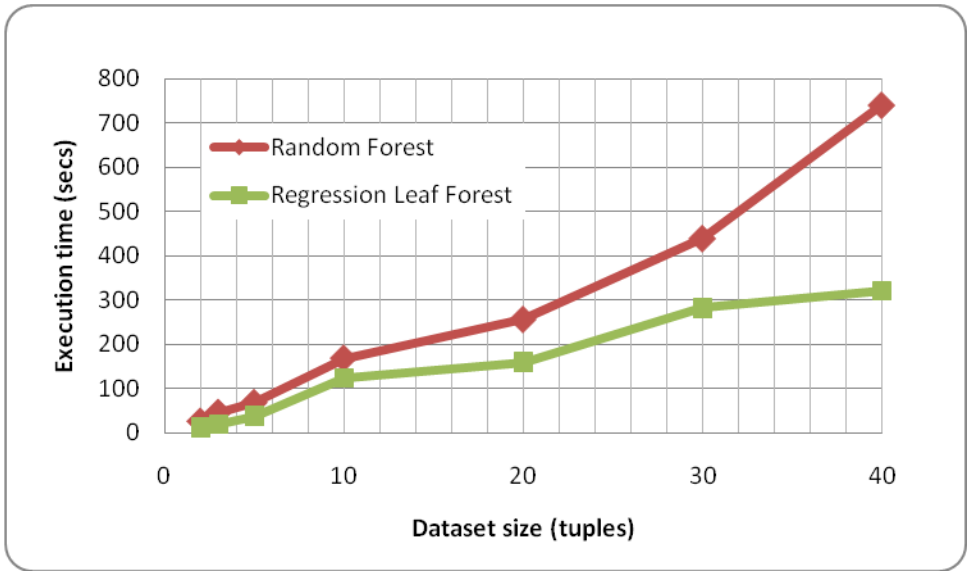

Figure 14: Performance of Regression Leaf Forest (lower plot) and Random Forest (upper plot) as number of dimensions increases when number of trees is 50.

The experimental result while varying dimensions (2 – 40) keeping fixed number of records (50,000) is described next. This was tested with 100 trees in the forest. Figure 15 shows the results.

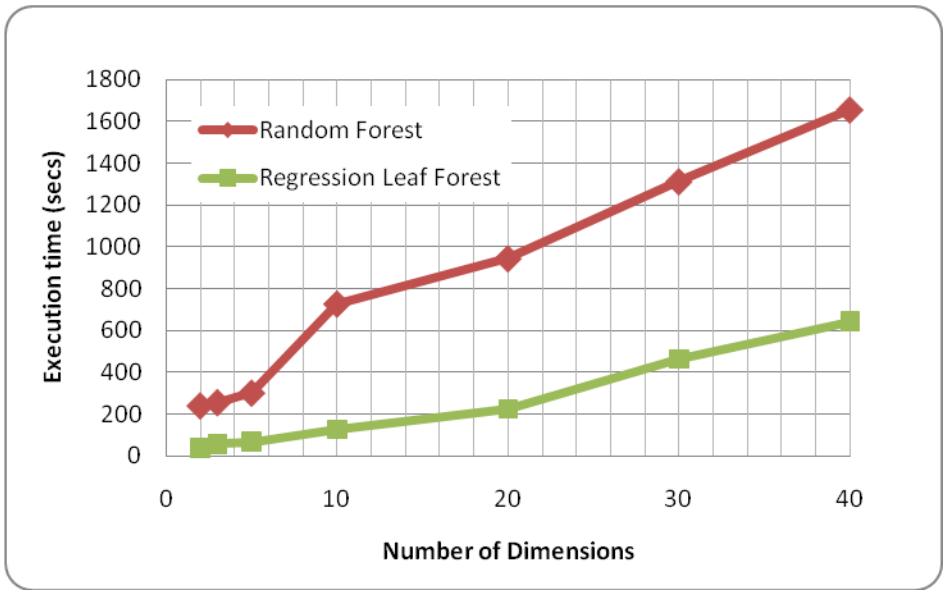

Figure 15: Performance of Regression Leaf Forest (lower plot) and Random Forest (upper plot) as number of dimensions increases when number of trees is 100.

From the above plots, it is clear that even with the increase in the number of dimensions in the dataset, Random forest with regression leaves (RLF) outperforms random forest with an increasing number of trees.

4.6 Experiment with Noisy and Missing data

Our second set of experiments considers both noisy and missing data. For this set of experiments we evaluate prediction accuracy. First, we will evaluate robustness; this is to evaluate accuracy as noise increases. The noise level varied from 5 percent to 40 percent and was generated as described previously. The prediction error was measured by the standard error estimates Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). We also computed the Mean Absolute Error (MAE) estimate to get how close our predictions are to the

eventual outcomes. The results are both tabulated and plotted in Table 2, and Y for MSE and RMSE respectively.

Noise %	Dimension	STLR (MSE)	RF (MSE)	RLF (MSE)	STLR (RMSE)	RF (RMSE)	RLF (RMSE)
5	20	1.1241	0.4154	0.2257	1.060235823	0.644515322	0.475078941
10	20	1.2459	0.5465	0.2678	1.116198907	0.739256383	0.517493961
15	20	1.9412	0.7945	0.2796	1.393269536	0.891347295	0.528772163
20	20	2.8412	0.9687	0.3597	1.685585952	0.984225584	0.599749948
30	20	4.2587	1.2457	0.5842	2.063661794	1.116109314	0.764329772
35	20	5.5741	1.5711	0.7521	2.360953197	1.25343528	0.867236992
40	20	7.4581	1.9778	0.9897	2.730952215	1.406342775	0.99483667

Table 2: Mean Squared Error and Root Mean Squared Error for varying Noise.

*STLR = Single Tree with LR

*RF = Random Forest

*RLF = Regression Leaf Forest

And the results of the experiments are plotted in the Figures 16 and 17 shows the result for MSE and RMSE respectively while varying the noise percentage from 5 to 40 with 10 trees in the forest.

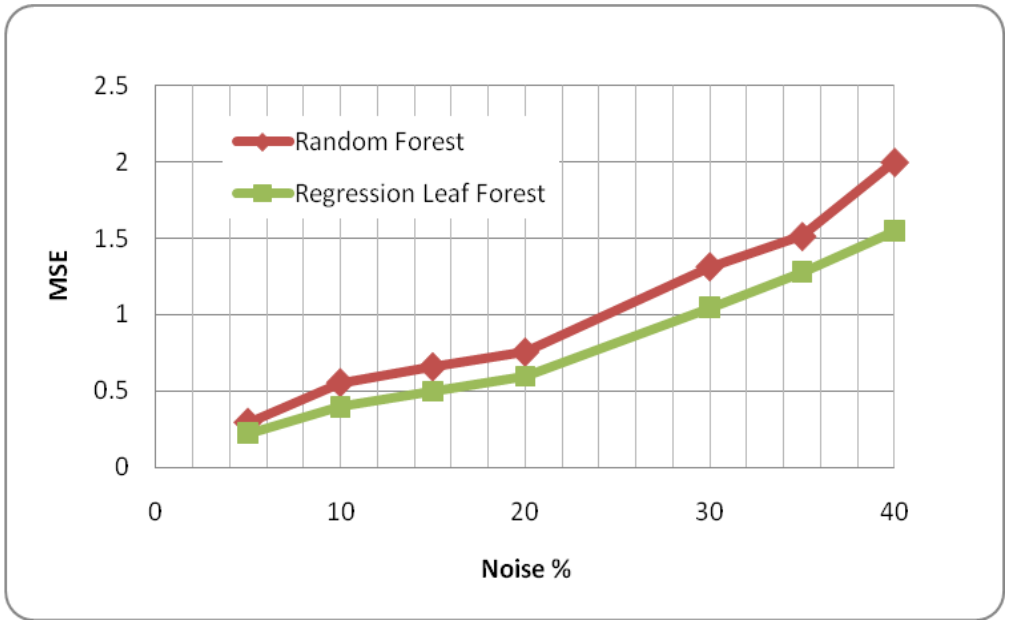

Figure 16: Mean Square Error estimate with increasing Noise between Random Forest (Upper plot) and Regression Leaf Forest (Lower plot) with 10 trees.

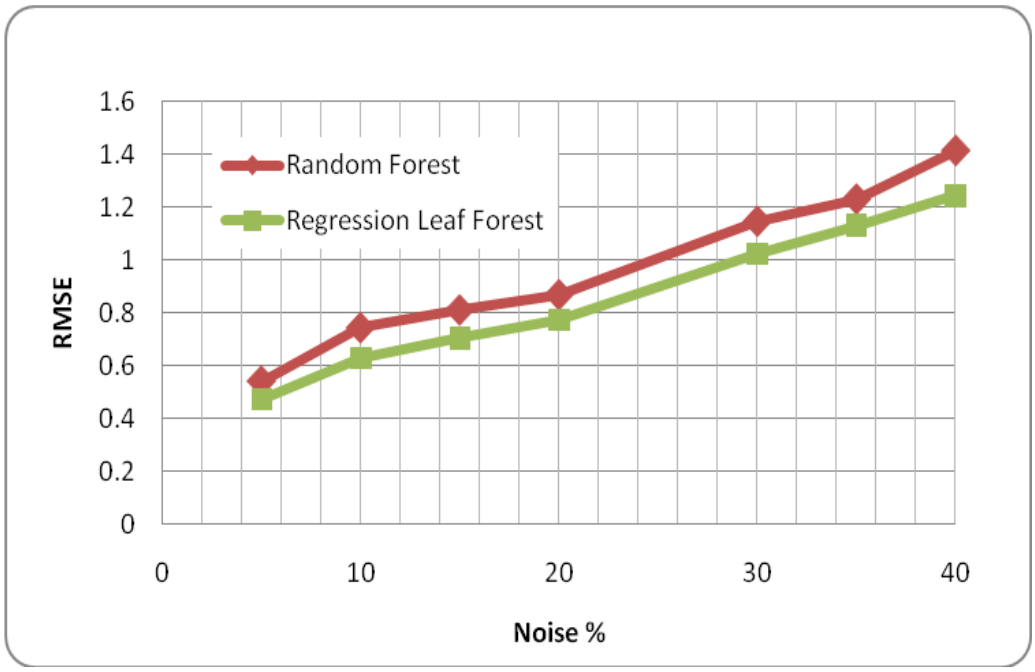

Figure 17: Root Mean Squared Error estimate with increasing Noise between Random Forest (Upper plot) and Regression Leaf Forest (Lower plot) with 10 trees.

The below plots are the result from the experiments described above; they show MSE, RMSE and MAE while varying the noise percentage from 5 to 40 with 100 trees in the forest**.** Figure 18, 19 and 20 shows the results of the error estimates. Here again we observe that Regression Leaf Forest outperforms Random forest and the performance advantage increases as noise percentage increase.

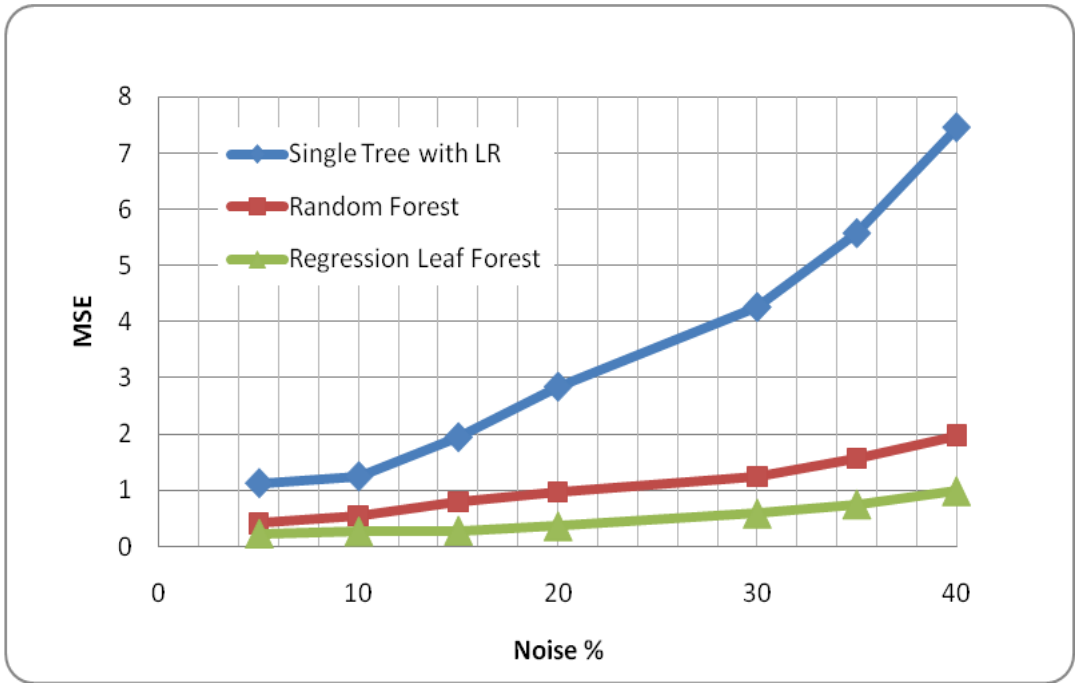

Figure 18: Mean Square Error estimate with increasing Noise between Single Tree with LR (Upper plot), Random Forest (Middle plot) and Regression Leaf Forest (Lower plot) with 100 trees.

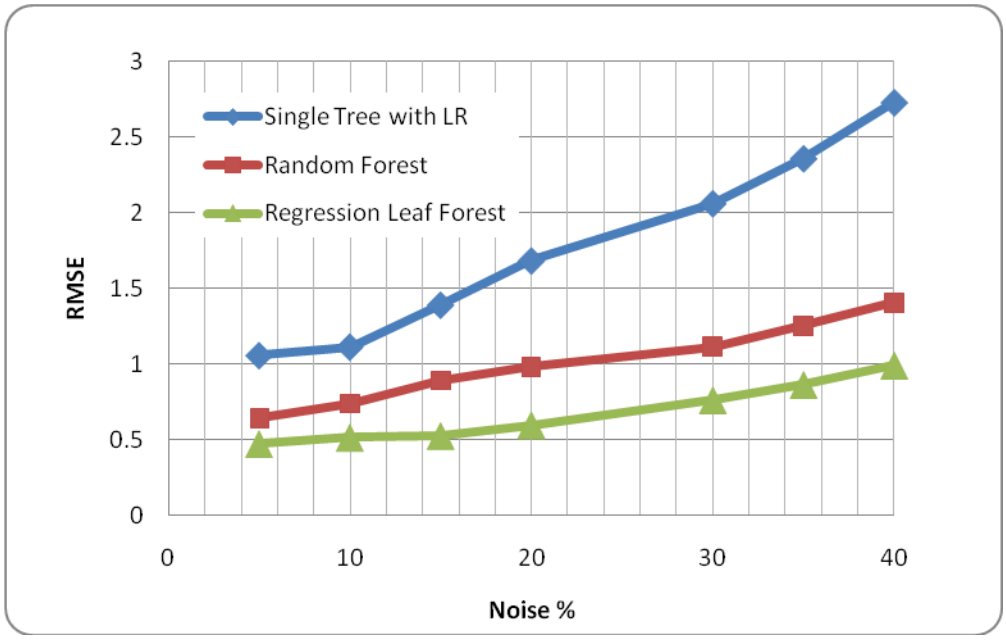

Figure 19: Root Mean Squared Error with increasing Noise between Single Tree with LR (Upper plot), Random Forest (Middle plot) and Regression Leaf Forest (Lower plot) with 100 trees.

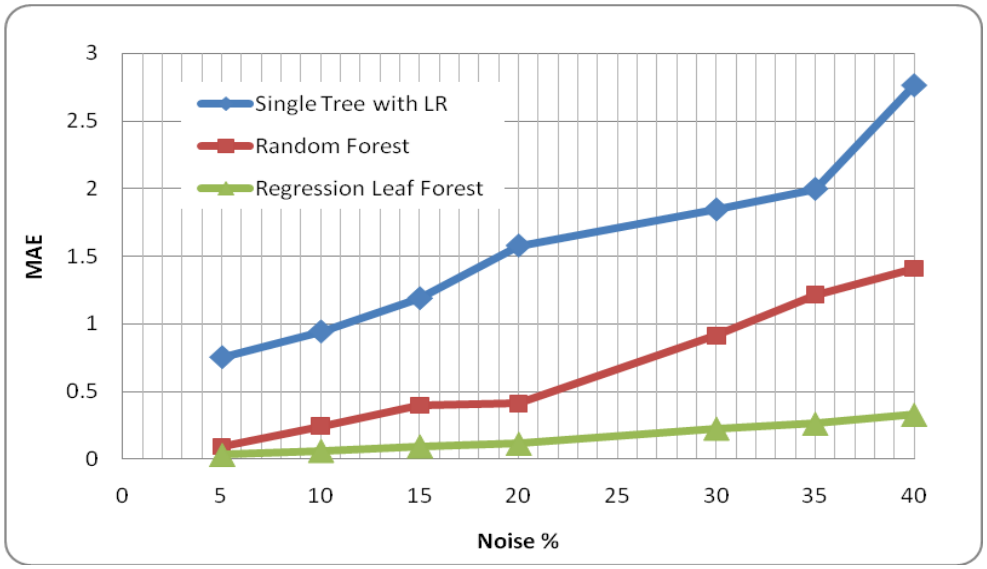

Figure 20: Mean Absolute Error with increasing Noise between Single Tree with LR (Upper plot), Random Forest (Middle plot) and Regression Leaf Forest (Lower plot) with 100 trees.

Here the above plots illustrate that our approach, RLF has better performance to Random forest, Single Tree (with LR) and never is less accurate.

4.7 Missing Data

Similar to the above experiments discussed, we varied the percentage of missing values in the dataset (5% to 35%) with minimal noise (10%) to find out the error estimates. Here for our experiments, we considered a synthetic dataset with 5,000 records.

The below plots shows the result for MSE, RMSE and MAE while varying the percentage of missing values from 5 to 35 with 100 trees in the forest. Figure 21, 22 and 23 shows the respective results.

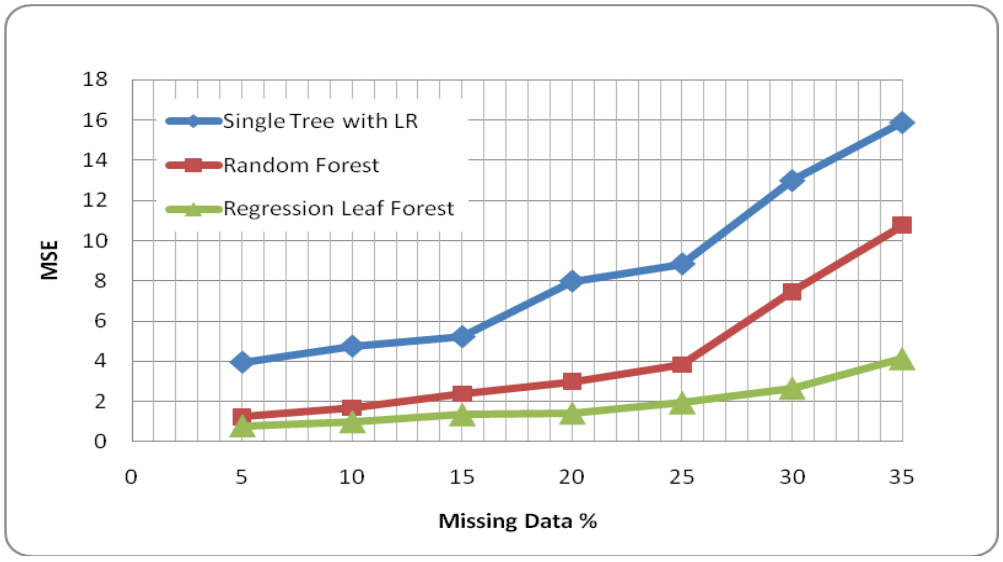

Figure 21: Mean Squared Error with increasing missing data between Single Tree with LR (Upper plot), Random Forest (Middle plot) and Regression Leaf Forest (Lower plot) with 100 trees.

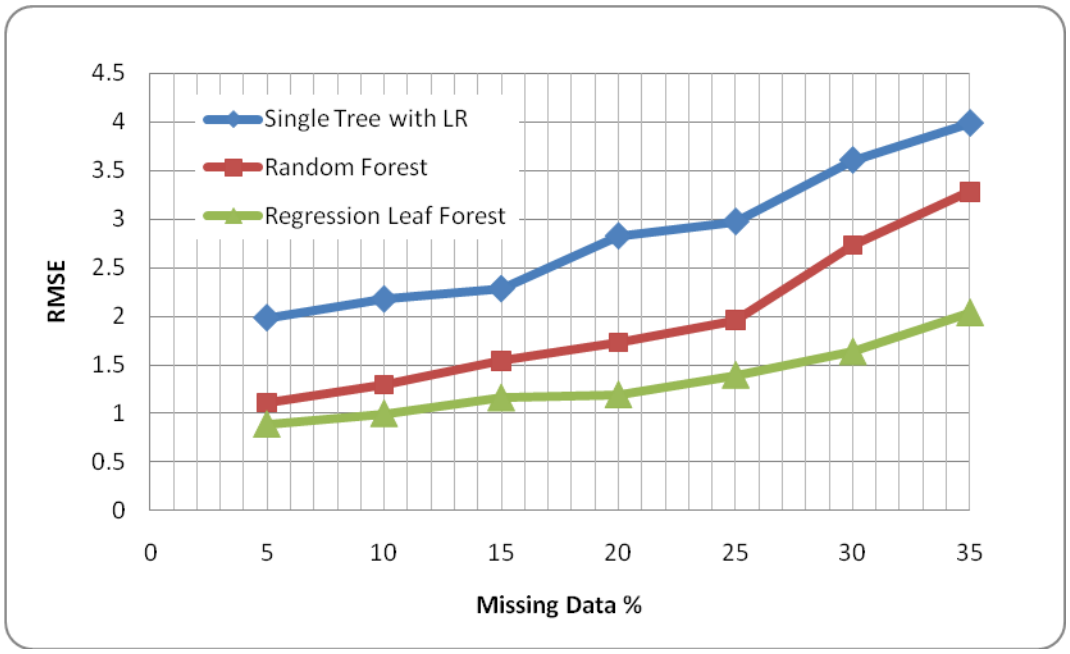

Figure 22: Root Mean Squared Error with increasing missing data between Single Tree with LR (Upper plot), Random Forest (Middle plot) and Regression Leaf Forest (Lower plot) with 100 trees.

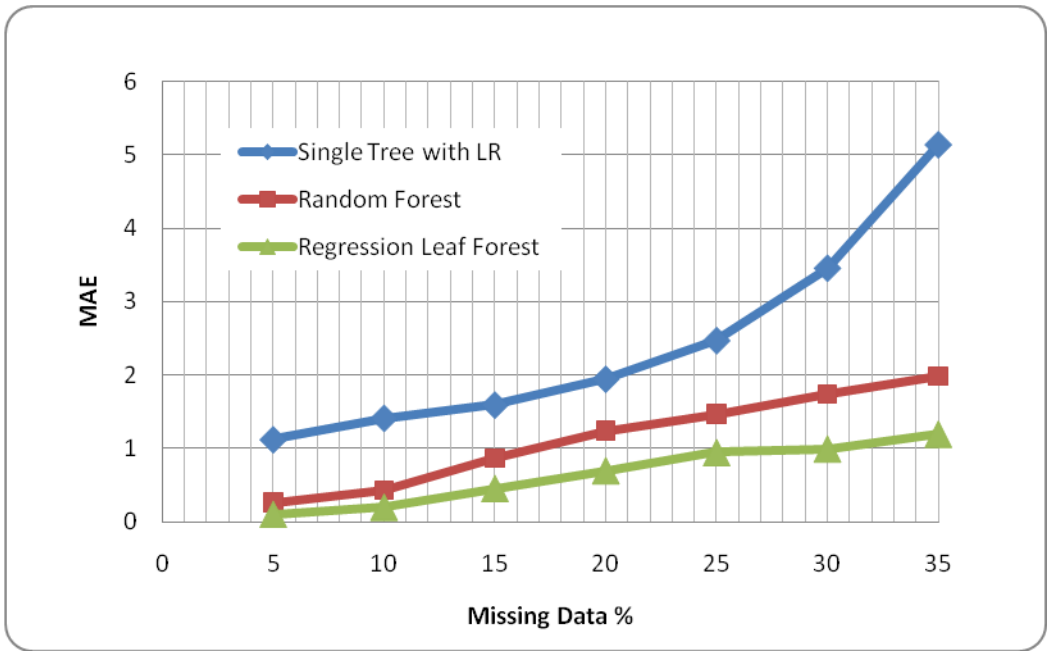

Figure 23: Mean Absolute Error with increasing missing data between Single Tree with LR (Upper plot), Random Forest (Middle plot) and Regression Leaf Forest (Lower plot) with 100 trees.

Once again the plots clearly illustrate that our approach, RLF has better performance to Random forest, Single Tree with LR. We can observe that Regression Leaf Forest outperforms Random forest and the performance advantage increases as noise percentage increase.

4.8 Experiment with benchmark datasets

The third set of experiments deals with real world data is described earlier in the section. We plot the results of our approach along with Single Tree, Single Tree with LR and Random Forest for MSE, RMSE and MAE. Each experiment used a 100 tree forest.

The first set shows the result of MSE with the real world datasets in Figure 24 and 25.

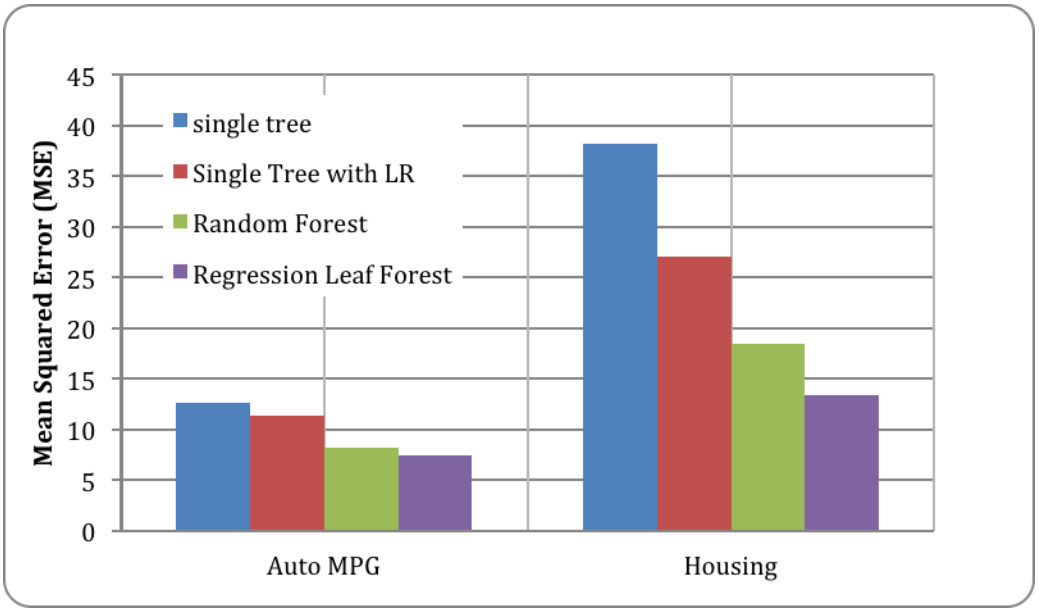

Figure 24: Mean Squared Error for AutoMPG and Housing datasets

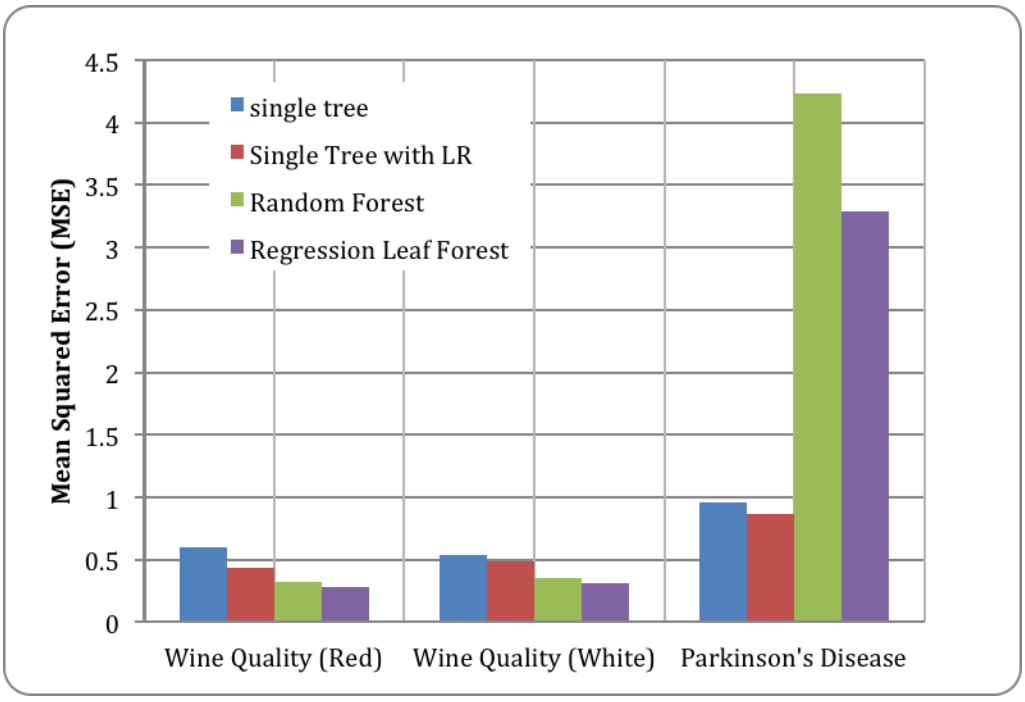

Figure 25: Mean Squared Error for Wine Quality (Red & White) and Parkinson's Disease datasets

The next set shows the result of RMSE and MAE in Figure 26 and 27.

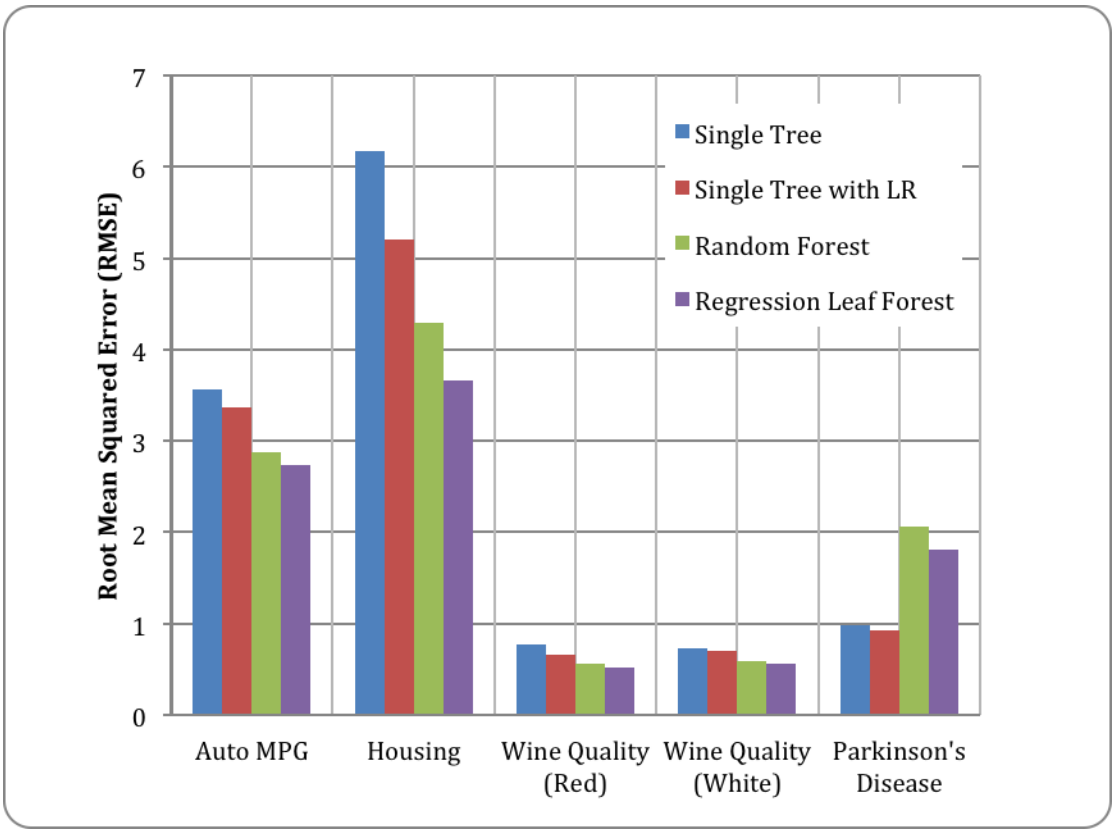

Figure 26: Root Mean Squared Error for AutoMPG, Housing, Wine Quality (Red & White) and Parkinson ’s disease datasets

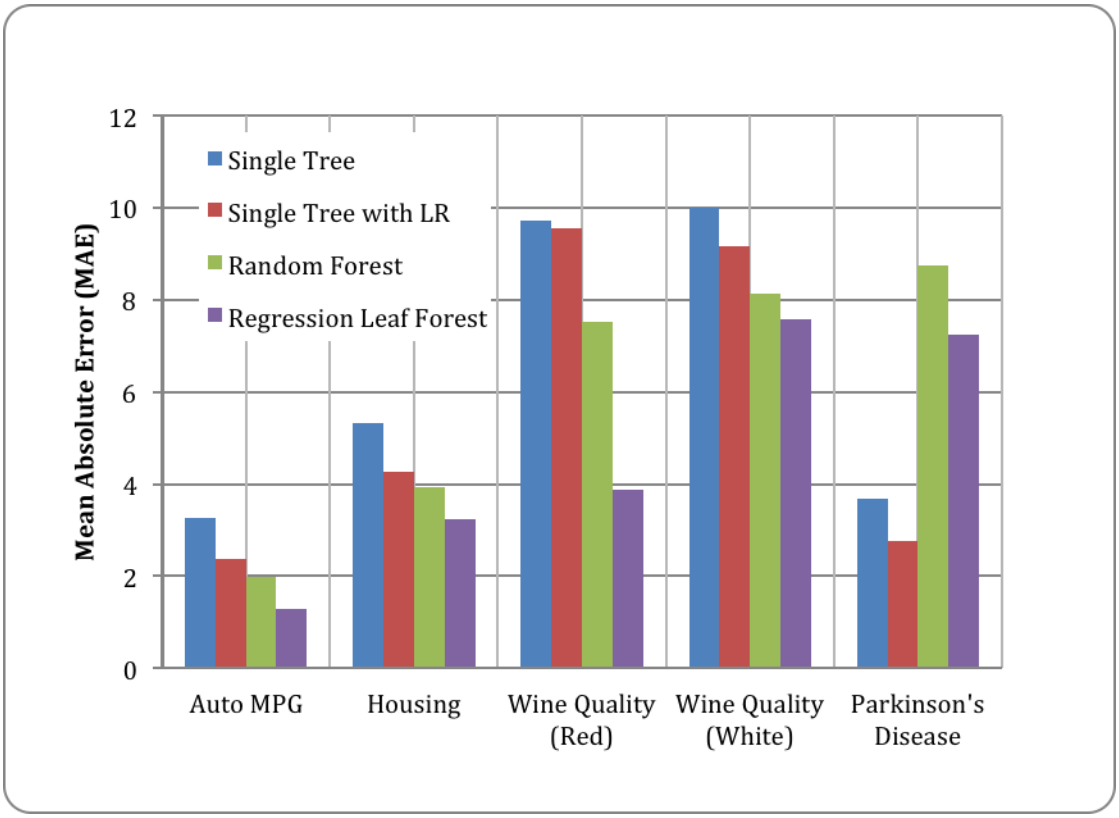

Figure 27: Mean Absolute Error for AutoMPG, Housing, Wine Quality (Red & White) and Parkinson's Disease datasets

From the above tables, Regression Leaf Forest showed consistently improved performance with respect to error estimates with various real world datasets. The only exception is in the Parkinson's disease data set where both the Single Tree algorithms perform better. A possible reason for RF to have less accuracy than singletree algorithms is that there may be only few good predictors among many non-informative predictors. The Parkinson's real world dataset had 24 attributes, out of which 16 were the required biomedical voice measures and others were non-informative description. Regression Leaf forest still performs better than random forests in all tests, and in all other tests (except for the Parkinson's disease data set, however, Regression Leaf Forest performs better both in terms of real time performance and accuracy. In future work

we will investigate further the limitations of Regression Leaf Forest but the results we have shown so far show strong promise that it is a viable candidate algorithm for large and high dimensional data sets.

CHAPTER 5

CONCLUSION

In this thesis we introduced a new hybrid approach for prediction problems with continuous data based on Random Forest and Linear Regression that effectively handles missing & noisy data and at the same time tends to show improved system performance. The main goal of this research is to create a system that can handle large data with multiple dimensions with minimum cost in performance. The approach here combines the regular random forest algorithm with a linear regression model. We do this by developing clusters at the very end of the tree instead of terminating it with the leaves. Then we apply linear regression model to these clusters and get our set of coefficients for the output function and finally we average the target for all the trees to get our desired target prediction. We have also introduced a wrapper class 'LrNaNed' to the linear regression model that handles the missing values (NaN's) in the data by adaptively replacing them with the medians.

We conducted extensive experiments on the performance and accuracy of the system. The experiments involved data sets both artificial and real world. We tested the system with large datasets and high dimensions; at the same time our tests introduced noisy and missing data to assess the predictive accuracy and performance. The results show that our approach is both effective and efficient when compared with the regular random forest and regression trees algorithms. In future we will further improve our system by adding different learning models instead of a single method, which in our intuition should further improve the predictive accuracy.

Also, the learning model may be extended such that it can deal with classification problems which are of great use in the real world.

BIBLIOGRAPHY

Berson, A., Smith, S., & Thearling, K. (1999). Building Data Mining Applications for CRM. McGraw-Hill.

Breiman, L. (2001). Random Forests. Machine Learning , 45 (1), 5-32.

Breiman, L., & Cutler, A. (n.d.). Random Forests Classification Description. Retrieved from http://www.stat.berkeley.edu/users/breiman/RandomForests/cc_home.htm

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Classification and Regression Trees.

Brown, G. (2004). Diversity in Neural Network Ensembles. The University of Birmingham.

Canlas, D. R. (2009, 5 August). DATA MINING IN HEALTHCARE: CURRENT APPLICATIONS AND ISSUES. MSIT .

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. Decision Support Systems , 47(4), 547-553.

Demmel, J., & Kahan, W. (1990). "Accurate singular values of bidiagonal matrices". Society for Industrial and Applied Mathematics. Journal on Scientific and Statistical Computing 11 (5) , 873–912.doi:10.1137/0911052.

Dietterich, T. (1998). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization. Machine Learning , pp. 1-22.

Dobra, A., & Gehrke, J. (2002). SECRET: A scalable linear regression tree algorithm. In Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 481–487). ACM.

Fausett, L. (1994). Fundamentals of Neural Networks: Architectures, Algorithms and Applications. New Jersey, USA: Prentice-Hall.

Fayyad, U., Grinstein, G., & Wierse, A. (2002). Information Visualization in Data Mining and Knowledge Discovery. Morgan Kaufmann Publishers.

Ferrari, G., & Muselli, M. (August 28-30, 2002). A New Learning Method for Piecewise Linear Regression. Proceedings of the International Conference on Artificial Neural Networks, (pp. 444-449).

Ferrari, G., & Muselli, M. (2002). A New Learning Method for Piecewise Linear Regression. Proceedings of the International Conference on Artificial Neural Networks, (pp. 444-449).

Friedman, J. (1988). Multivariate Adaptive Regression Splines. Technical Report 102. Stanford University, Laboratory for Computational Statistics, Stanford CA.

Hartmann, M. (1997). Signals, Sound, and Sensation. Springer. (ISBN 1563962837).

Hegger, R., & Kantz, H. (1999). Improved false nearest neighbor method to detect determinism in time series. Phys Rev E (60:4970–3).

Ishwaran, H., Kogalur, U., Blackstone, E., & Lauer, M. (2008). Random survival forests. Ann Appl Stat 2 , 841-860.

Kamber, J. H. (2000). Data Mining: Concepts and Techniques. Morgon Kaufmann Publishers.

Karalic, A. (1992). Linear regression in regression tree leaves. In International School for Synthesis of Expert Knowledge .

Loh, W.-Y. (2002). Regression trees with unbiased variable selection and interaction detection. Statist. Sinica 12 , MR1902715, 361-386.

Meinshausen, N. (2009). Node Harvest: simple and interpretable regression and classification. arXiv:0910.2145.

Mitchell, T. (1997). Machine Learning. McGraw-Hill Science/Engineering/Math, 1st edition.

Mitchell, T. (1997). Machine Learning. McGraw-Hill Science/Engineering/Math 1st edition.

Qi, Y., Klein-Seetharaman, J., & Bar-Joseph, Z. (2005). Random forest similarity for protein-protein interaction prediction from multiple sources. Pac. Symp. Biocomput. 10, (pp. 531-542).

Quinlan, J. R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning (pp. 236-243). University of Massachusetts, Amherst: Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of Decision Trees. In J. R. Quinlan, & J. W. Dietterich (Ed.), Readings in Machine Learning (pp. 81-106). Morgan Kaufmann.

Quinlan, J. R. (1992). Learning with Continuous Classes. 5th Australian Joint Conference on Artificial Intelligence, (pp. 343-348). Singapore.

Scott, D. W. (1979). On optimal and data-based histograms. Biometrika , 66, 605–610.

Shinomoto, S. (2007). A method for selecting the bin size of a time histogram. Neural Computation , 19, 1503-1527.

Tao, Q. (2004). MAKING EFFICIENT LEARNING ALGORITHMS WITH EXPONENTIALLY MANY FEATURES. Qingping Tao - A DISSERTATION Faculty of The Graduate College University of Nebraska In Partial Fulfillment of Requirements .

Vogel, D., Asparouhov, O., & Scheffer, T. (2007). Scalable look-ahead linear regression trees. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. , 757-764.

Wolpert, D. a. (1999). An Efficient Method to Estimate Bagging's Generalization Error. Machine Learning , 35, 41-55.

APPENDIX A

IMPLEMENTATION PSEUDOCODE

Binning procedure

```
1   //define bin boundaries to split the data
2   def binboundaries(l,h,s):
3      #boundary = []
4      …
5      for i in np.arange(l, h, s):
6         boundary.append(i)
7         #print boundary
8      return boundary
9
10  //discretize the data to accommodate into the bins
11  def discretedata(array list, bins):
12     ...
13     for i in range(len(d)):
14              for j in range(len(b)):
15           if(int(d[i] > b[j])):
16
17           elif (int(d[i] < b[j])):
18                    avg = (b[j]+b[j-1])/2
19              newval.append(avg)
20              Break
21  …
22  return arraylist
```

Figure A.1: Binning and Discretizing the data.

The figure above shows the binning pseudocode. This is the preprocessing step before beginning to build the trees.

BootStrap Procedure

```
1   //Bootstrapping process for random sampling with replacements
2   def bootstrap(n, newx1, newx2, newy):
3     ...
4   #define sampling data variables
5   my_data = []
6   #generate bootstrap data
7      for i in range(len(newx1)):
8         r = random.randrange(1,n)
9         ...
10  #Append to bootstrap list
11  append(newlist[i])
12  .....
13  //write a new list which is our bootstrap
14     for item in range(len(bootlist)):
15        #write(//file);
16    append(bootstlist[item])
17  ...
18  END
```

Figure A.2: BootStrap Pseudocode

The figure above shows the Bootstrap pseudocode. This is the initial stage of our tree building procedure where we perform the bootstrap aggregation.

Forest Builder

```
1   //initial splitting of data set
2   [traintargets,trainqueries]=readtargets(INPUTDATA)
3   [testtargets,testqueries]=readtargets(TESTDATA)
4   targets=traintargets+testtargets;
5   ...
6   for itr in range(0,ITER):
7      # write target
8      for i in range(0,ntra):
9   …
10     # read prediction
11     for i in range(0,len(totalpreds)):
```

```
    l=p.stdout.readline()
    try:
 totalpreds[i] += float(l.split(' ',1)[0])
    except:
sys.exit(1)
    # get and store results
...
def forestbuilder(traindata, testdata, tcount, train_preds, test_preds, args):
  int trees = tcount;
  for t in range (0, trees):
    single_forest(train, test, train_preds, test_preds, args);
  // average results of the trees
  for  i in range (0, trees):
    for j in range (0, test):
      test_preds[i][j] /= trees;

//random forest results
  for  i in range (0, test):
    predict_result(test_preds[i], test[i], args.test_outs[i]);
}
}
def single_tree(train, test, train_preds, test_preds, args):
data_t sample
randsample(train, sample)
...
for i in range(num_test):
double rmse = classify_all(test[i], tree, test_preds[i], args)
double rmse = classify_all(train, tree, train_preds, args)
....
##Cluster the datapoints using args
if(ith node in tree is at cluster point dp)
clusterify(tnode_[i] till the leaf)
....
return cluster[i];
...
  #delete tree;
...
}
```

Figure A.3: Forest Builder Pseudocode

The figure above shows the pseudocode for forest building procedure. In this stage we grow multiple trees from the bootstrap data as per the random forest algorithm.

LrNaNed - Wrapper class to LReg Model

```
1     Process the nodes of the trees till cluster is reached.
2     ...
3     for (clustered attributes in data):
4         do Check for NaN's
5             if (NaN_FOUND)
6                 missing_list = compute_getMedians(train_list, key_index)
7                 Get the medians replaced in new list
8                 replaced_list = replace_NaN(missing_list, key_index)
9                LReg_compute(missing_list)
10            elif (NO_NAN)
11            #cluster of data points
12           LReg_compute(unreplaced_list)
13    ...
14    END
```

LReg Model – Linear Regression model

```
1     LReg_compute(n_list of Cluster, tree):
2           Get the Kth cluster for Kth tree in the forest
3           calculate LR for the cluster[k]
4     ...
5           def normalize(data,attrmean,attrstd):
6           # transform all variables so that they all have 0 mean and 1 stdev
7           # for every column/attribute
8           for j in range(0,shape(data)[1]):
9          # replace entire old column vector with normalized data
10         data[:,j]=(data[:,j]-attrmean[j])/attrstd[j]
11        result=linalg.lstsq(X_train,Y_train)
12        #Get the sum of residuals
13        #Compute the Errors
14        return data
15    ...
```

```
16              predict_target(cluster[k] list)
17          ...
18          END
```

Figure A.4: LrNaNed and LReg Pseudocode

The figure above shows the pseudocode for our novel approach that introduces wrapper class ‘LrNaNed’ to the clusters before applying the linear regression model. This process handles missing data.