Department of Computer Science

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Sampling-based Prediction of Algorithm Runtime

Quan Sun

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science at The University of Waikato.

September 2009

Abstract

The ability to handle and analyse massive amounts of data has been progressively improved during the last decade with the growth of computing power and the opening up of the Internet era. Nowadays, machine learning algorithms have been widely applied in various fields of engineering sciences and in real world applications. However, currently, users of machine learning algorithms do not usually receive feedback on when a given algorithm will have finished building a model for a particular data set. While in theory such estimation can be obtained by asymptotic performance analysis, the complexity of machine learning algorithms means theoretical asymptotic performance analysis can be a very difficult task. This work has two goals. The first goal is to investigate how to use sampling-based techniques to predict the running time of a machine learning algorithm training on a particular data set. The second goal is to empirically evaluate a set of sampling-based running time prediction methods. Experimental results show that, with some care in the sampling stage, application of appropriate transformations on the running time observations followed by the use of suitable curve fitting algorithms makes it possible to obtain useful average-case running time predictions and an approximate time function for a given machine learning algorithm building a model on a particular data set. There are 41 WEKA (Witten & Frank, 2005) machine learning algorithms are used for the experiments.

Acknowledgments

Foremost I would like to thank my supervisor Associate Professor Eibe Frank for his help and generous support during this project. He contributed many of the ideas in this thesis. Moreover, he proofread my thesis and commented on every aspect of it. Without his constant guidance and encouragement, I would not have been able to finish this thesis.

During the period of my study, the University of Waikato has provided me with much appreciated financial support through a Master's research scholarship, and the Department of Computer Science also offered me a graduate assistant position. I would like to thank all the people in the Department of Computer Science, especially the members of the machine learning laboratory. I also would like to thank John Moriarty for proofreading the draft of the thesis.

A special thank you to my wife Meng Wang, who was always there for me, even though she is in the final year of her doctorate. I know that cooking three meals a day is harder than writing a thesis. Thank God for blessing me with you. Thank you, Meng.

Finally, I would like to thank my mother for endless support and love, and making my education in New Zealand possible. You will never know how much your encouragement meant to me. This thesis is dedicated to you.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning algorithms have been adopted in many real world applications. However, there is an issue when applying machine learning in practice: users generally do not know when the algorithm will have finished building a model on a given training data set. Much time may be wasted waiting for an algorithm to finish, especially when the training data set is very large. Thus, the ability to predict a machine learning algorithm's running time could be very useful.

Two kinds of approaches can be used to estimate the running time of an algorithm. The first approach works when the target algorithm is simple. It uses knowledge about the underlying algorithm to perform a theoretical performance analysis, and then uses this information to estimate the running time. For more complex algorithms, such as machine learning algorithms, such an approach can be very difficult. In these cases another kind of approach, called "empirical algorithm analysis", is more useful. Empirical algorithm analysis employs sampling-based techniques to construct a function that is an approximation to the true running time function of a given algorithm.

The present work focuses on the latter approach, and has two goals. The first goal is to implement sampling-based running time estimators that can predict the running time of a machine learning algorithm building a model on a given training data set. The second goal is to use experiments to evaluate the prediction performance of these estimators.

1.1 Basic definition of estimation problem

Let f and g be two functions of a natural number n . If f and g are asymptotically equivalent as $n \to \infty$ then

$$\lim_{(n\to\infty)} \frac{f(n)}{g(n)} = 1.$$

Assume $f(n)$ is the running time function of an algorithm, where n is the input size. The goal of this work is to find a method that can automatically construct an estimation model (asymptotic function) $g(n)$ based on sampling techniques, and then use this model to predict the running time t where $t = f(n)$.

1.2 Applications of sampling-based prediction of algorithm running time

The direct application of running time estimators is the domain of running time prediction for an algorithm based on a given input size. Algorithm users can run an estimator before the algorithm proceeds to the full task. The estimated running time value may help users get a general feeling of how much time the algorithm will require to complete a given task. This is the initial motivation for this work.

Another application of the estimators proposed in this work is to provide users with an asymptotic function (in most cases it is a trend function) of the true running time function. The form of the estimated asymptotic function can be very useful in the domain of improving data mining utility with projective sampling, which is a new and emerging research area. The idea of projective sampling (Last, 2009) is to fit a function to a partial learning curve obtained from a small subset of potentially available data, and then use it to analytically estimate the optimal training set size for a machine learning problem. The authors in (Last, 2009) assume the total cost of a machine learning algorithm

induced from n training examples can be calculated by the following expression

$$TotalCost(n) = n \cdot C_{tr} + err(n) \cdot |S| \cdot C_{err} CPU(n) \cdot C_{time}, \qquad (1.1)$$

where S is the testing set, C_{tr} is the cost for acquiring each new training example, C_{time} is one unit of CPU time, C_{err} is the cost for each misclassified example from the testing set, $CPU(n)$ stands for the time required to induce a classification model from n examples and $err(n)$ is the error rate of this model. The projective sampling algorithm presented in (Last, 2009) aims at defining a heuristic sampling strategy P^* that minimizes the total cost of a classification process

$$P^* = \underset{P}{\operatorname{argmin}}\ TotalCost(P).$$

To make (1.1) as accurate as possible, we need to precisely calculate each of its terms. We here focus only on the term $CPU(n)$. The authors used two simple methods to obtain an approximation to $CPU(n)$: simple linear regression and the power law method (both are discussed in Chapter 2).

The authors claim that model (1.1) works very well in terms of approximating the total cost for a classification problem. However, our experimental results show that the two estimators for $CPU(n)$ they consider are not sophisticated enough when used to approximate the running time function of a machine learning algorithm in general. The estimators proposed in this thesis may improve the accuracy of the estimate for $CPU(n)$, and thus the accuracy of model (1.1).

There are many other application domains for running time prediction. For instance, the ideas and techniques employed by running time estimators can be modified and easily adapted for predicting other resources required by an algorithm, such as the memory requirement of a given input size.

1.3 Objectives and thesis overview

A literature review shows that there are not many publications focusing on the running time prediction problem. Existing research in this area is mainly based on using simple linear regression or the power rule method as a tool for interpolation problems. Moreover, even the simplest running time prediction methods have not been thoroughly evaluated against each other. Although some advanced regression methods are available, most of them were designed and investigated in the field of time series analysis or for a very specific problem.

When this project was started, the initial assumption was that simple linear regression would work very well. However, it turned out very quickly that while this is a theoretically simple problem, it is practically rather challenging. Those challenges are addressed in this thesis, through discussion of the following questions:

- How should the running time of an algorithm be measured?
- Do sophisticated point estimation techniques improve data quality?
- How should running time data points be sampled?
- How ought interpolation techniques for extrapolation problems be extended?
- How can one fairly and systematically compare the prediction performance of running time estimators in terms of using different evaluation strategies?

These questions resulted in a detailed investigation of the running time problem that consisted of three stages: sampling, model construction and evaluation.

The remainder of this thesis is structured as follows. The next chapter is devoted to giving a detailed introduction to the running time prediction problem, and focuses on theoretical considerations regarding several curve fitting

algorithms that can potentially be employed as an extrapolation tool. The chapter that follows next (Chapter 3) elaborates on the eleven running time estimators proposed in this work, and how they are constructed. Chapter 4 discusses the methods for running time measurement, and point estimation techniques used to improve the quality of observed running time data. Chapter 5 contains an empirical evaluation of the running time estimators in this thesis on a wide range of running time data sets obtained by monitoring 41 WEKA machine learning algorithms. Chapter 6 contains the conclusion of this thesis and a summary that briefly describes future work. There are some more implementation and mathematical details in the appendices, which are referenced when necessary.

The work presented in this thesis builds on work done for a one-paper dissertation by the same author (Sun, 2008). It extends it these ways:

- additional curve fitting algorithms are discussed in depth regarding both theoretical and practical aspects (least absolute deviations and non-negative least-squares);
- running time estimators are proposed and examined empirically (LsF, LsR, LsSeq, LadF, LadR, nnlsF, nnlsR and OneTest);
- four additional running time measurement methods are introduced and compared against each other;
- evaluation by examining the quality of each prediction is introduced as a new method for evaluating predictive performance of the estimators;
- evaluation results are presented that are based on real world data sets;
- estimation of data mining utility is discussed as an application of running time prediction.

Description of techniques that can be found in (Sun, 2008) have been reused in Chapters 2 and 3 of this thesis where appropriate.

Chapter 2

Background

An algorithm, in terms of computer science, is a step-by-step procedure for solving a given task with a finite amount of resources, such as time, storage and computer memory usage. A given task may potentially be completed using different algorithms with different sets of resource requirements. One goal of algorithm study is to find methods that can be used to precisely calculate—or at least approximate—how much of a particular resource is required for a given algorithm on a particular task. Such study is referred to as algorithm analysis.

In this work, we investigate methods that can be used to predict the running time of machine learning algorithms, and evaluate these methods empirically by experiments. We generally focus on asymptotic analysis of an algorithm, which formally can be thought of as a method of describing limiting behavior (see definition in Section 1.1).

In the subsequent sections of this chapter, we focus on background information regarding both theoretical and practical aspects of algorithm analysis.

2.1 Deductive and inductive approaches

From the perspective of scientific methods, approaches to algorithm analysis can be categorized as either deductive or inductive approaches. The asymptotic behavior of an algorithm can be deduced from strong hypotheses or induced from experiments.

The deductive approach works from the more general to the more specific, and is sometimes informally referred to as a top-down approach (Trochim,

2006). In the context of predicting the running time of an algorithm we assume that there is an algebraic relation between the size of input and the running time of a given algorithm. All other factors that may contribute to variations in running time are treated as random noise. In this way, an observation is defined as a $\{n, t\}$ pair, where n is the size of input, and t is the running time; and a theory is defined as a function that precisely maps n to t for each $\{n, t\}$ pair in the observations. A deductive approach begins with formulating a function g expressing the relation between the size of input n and the running time t of an algorithm. The function g can be seen as a hypothesis, and is examined by actual observations. In our context, the input sizes and observed running time—the $\{n, t\}$ pairs—are collected to examine the hypothesis, i.e. function g. This enables one to draw a conclusion on whether the hypothesis (original theory) is confirmed, or not. The inductive approach works the other way, moving from specific observations to generalizations. This is informally called a bottom-up approach. In the context of this work, an inductive approach begins with collecting observations, the $\{n, t\}$ pairs, and then detecting patterns, or regularities, in order to develop a function g that can precisely map each $\{n, t\}$ pair in the observations. In this work, we investigate sampling-based running time estimation methods for machine learning algorithms. This can be seen as an inductive approach.

2.2 What is sampling-based prediction?

The underlying idea of sampling-based prediction of algorithm runtime can be seen from Figure 2.1. For instance, we want to have an estimate for the running time t of algorithm A for a given input data set X, of size n. In the sampling stage, we observe the running times of A for inputs that are k sub sets of X. The input size of each sub set is n_i. Suppose $k = 3$; then for each run-instance the running time and the input size are observed. Now, we have three observations: $Z_1 < n_1, t_1 >$, $Z_2 < n_2, t_2 >$ and $Z_3 < n_3, t_3 >$, where

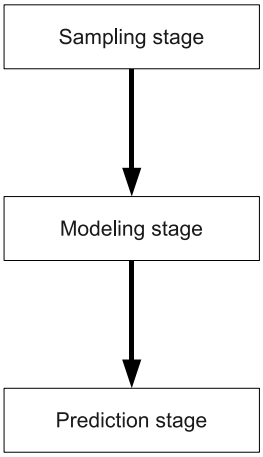


Figure 2.1: A flow chart showing the concept structure of sampling-based prediction of algorithm running time

the t's are the running times, and the n's are the input sizes, subject to the restriction that $n_1 < n_2 < n_3 < n$. At this point, we say that we have a *training* data set with three sample data points. Then, in the modeling stage, we choose a mathematical method, usually a regression method, to build a model for the training data set. Finally, an estimate for the running time of A on X is given by the mathematical model in the prediction stage.

2.3 Why use machine learning algorithms?

Machine learning research is concerned with the question of how to construct computer algorithms that automatically improve with experience (Mitchell, 1997). It draws on concepts and results from many fields, including mathematics, statistics, philosophy, biology, control theory and information theory. Because of the interdisciplinary nature of machine learning, its algorithms usually are very complex and need to be able to handle a large amount of data. When doing a machine learning experiment for a particular algorithm and data set, the researcher may have to wait for hours, or even days, to get the results. This motivated us to investigate how to predict the running time of an algorithm, particularly for machine learning algorithms. In this work, WEKA (Witten & Frank, 2005) is employed as the framework for constructing and

programming the sampling-based running time estimators. Also, the prediction performance of the estimators is examined by predicting the running time for 41 machine learning algorithms written in WEKA. A list of names of those algorithms can be found in Appendix B.

2.4 Empirical asymptotic analysis

As we have mentioned in Section 2.1, there are two kinds of approaches that can be used to estimate the running time of an algorithm. One kind of approach is to use knowledge about the underlying algorithm to perform a theoretical performance analysis, and then use this information to estimate the running time. This approach works when the target algorithm is simple. For more complex algorithms, such as machine learning algorithms, applying this approach can be a very difficult task. Another kind of approach is referred to as empirical algorithm analysis, and employs sampling-based techniques to construct a function that is an approximation to the true running time function of a given algorithm.

In this work, we focus on the latter approach. In the following sections, basic ideas of empirical algorithm analysis and numerical function approximation approaches for both interpolation and extrapolation are discussed, with examples of finding closed form expressions for the running time of machine learning algorithms, in terms of input parameters of interest—the input size of a particular algorithm run instance.

2.4.1 Interpolation curve fitting

The underlying patterns of many practical problems can be described mathematically by a function $y = f(x)$. In such a case, we may face two situations. One is that our knowledge of the problem is limited, so we do not know the analytical form for the problem. All we can do is obtain values for certain data points from experiments. Another situation is where we know the analytical

form of $f(x)$, but it is too complex to be applied directly. For this reason, we need to find a proper function $P(x)$ as an approximate function to the original function $f(x)$. Interpolation can be employed to solve such problems.

Interpolation is a method or procedure of constructing functions within the range of a discrete set of known data points (Li *et al.* , 2000). More precisely, given a sequence of n distinct numbers x_k, called nodes, and, for each x_k, a second number y_k, we are looking for a function P so that

$$P(x_k) = y_k,\ k = 1, \ldots, n.$$

A pair x_k, y_k is called a data point and P is called an interpolant for the data points. There are many forms of interpolation methods, such as linear interpolation, polynomial interpolation, spline interpolation, interpolation via Gaussian processes and others (Li *et al.* , 2000). Here is a simple example: assuming we have the following data:

1. $x_1 = 1, y_1 = 2,$

2. $x_2 = 2, y_2 = 3,$

3. $x_3 = 4, y_3 = 6;$

If we want to know the value of y given $x_{new} = 3$, then this is an interpolation problem, since the data point to be predicted is in the range of the known data points ($1 < x_{new} < 4$).

It is clear that the mechanism of interpolation is not directly applicable to the running time prediction problem, since we are interested in predicting a data point beyond the range of the observed data points. However, some particular interpolation methods can be extended to be applicable to an extrapolation problem, e.g. curve fitting methods.

In the context of this work, the goal is to use data points obtained in the sampling stage to form a mathematical model that can be used as an

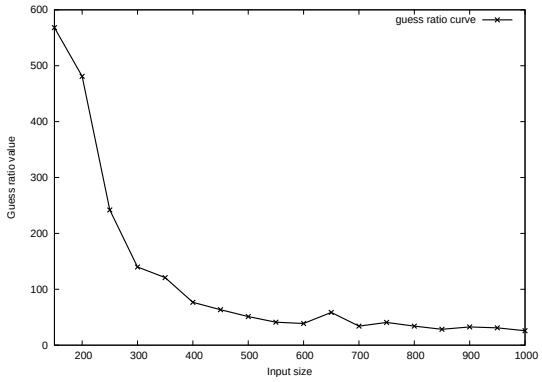
estimator for unknown data points. Naturally, if a curve fits the observed running time data points well, it is a good candidate for a proper running time estimator. In the following sections, we introduce some curve fitting methods for interpolation. They all serve the purpose of fitting the observed running time data points, and have predictive potential. We consider the guess ratio method, the guess difference method, the power rule method, the Box-Cox transformation method, the ladder transformation method, simple linear regression, multiple linear regression, least absolute deviations regression and the non-negative least-squares methods.

2.4.2 Guess ratio test

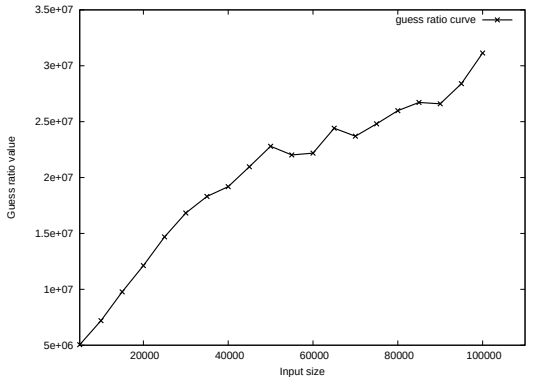
The underlying idea of the guess ratio test is to assume the main term of an algorithm's running time function can be formulated by $g(n) = n^c, c > 0$, where n is the input size.

Under this assumption, let $t(n)$ denote the observed running time. In (McGeoch *et al.* , 2002), the guess ratio $r(n)$ is defined as $\frac{t(n)}{g(n)}$. If the ratio grows as the input size increases, then $g(n)$ underestimates the running time; if the ratio converges to 0 as the input size increases, then $g(n)$ is an overestimate. In the case that the ratio converges to some constant b greater than 0, then $g(n)$ is a good estimate for the growth rate of $t(n)$. In addition, an estimation model for predicting the running time of unobserved input sizes can be constructed based on the best guess function after several guess ratio tests by using b as an estimated of c. In practice, empirical study can test only a finite number of input sizes. Therefore, the guess ratio test cannot be guaranteed to find the exact value of the exponent c with a finite number of input sizes.

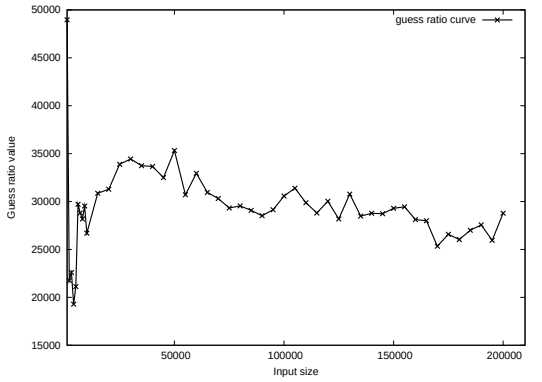
The following example uses the guess ratio test to find an appropriate estimate of the running time bound of a decision tree algorithm. It is claimed that the cost of constructing a J48 decision tree (WEKA implementation of the classic C4.5 decision tree) without sub tree raising is $O(mnlogn)$ where m is the number of the attributes, and n is the number of training examples for



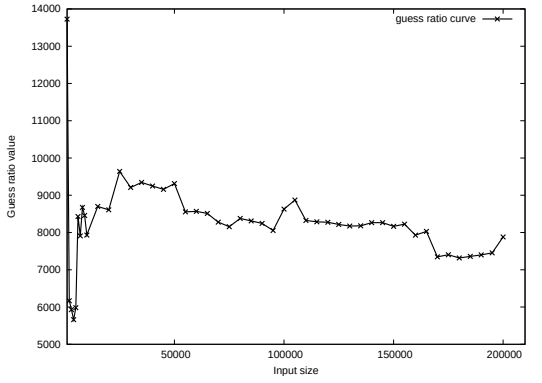
(a) $g(n) = n^2$



(b) $g(n) = n^{0.5}$



(c) $g(n) = n^{1.1}$



(d) $g(n) = nlog(n)$

Figure 2.2: Guess ratio curves for different ratio tests

the J48 decision tree algorithm. In most cases, the number of attributes m is less than the training instance size n; thus m can be ignored, and n can be seen as the input size. In this case, using big-Oh (Goodrich & Tamassia, 2002) notation, the simplest running bound form of J48 is $O(nlogn)$.

Next, we apply the guess ratio test, and assume the bound function of the J48 algorithm can be formulated by $g(n) = n^c, c > 0$. Our first experiment begins with guessing $g(n) = n^2$ because it is safe to say $O(nlogn)$ is $O(n^2)$. To obtain running times, J48 was run on a data set consisting of three nominal attributes, six numeric attributes and one class attribute. Figure 2.2 (a) shows that the ratio converges to close to 0 for $n = 1000$. Hence, by applying the concept of the guess ratio test, we conclude that $g(n) = n^2$ is an overestimate of $t(n)$. That is true, because in this case $O(t(n))$ is $O(nlog(n))$.

The next experiment is designed to see whether $g(n) = n^{0.5}$ is a good estimate, in this case with $n \leq 100000$. Figure 2.2 (b) shows that the ratio grows as the input size increases, therefore $g(n) = n^{0.5}$ underestimates the running time. At this stage, the guess ratio test results suggest that c should be between 0.5 and 2.0.

The next experiment is designed to see whether $g(n) = n^{1.1}$ is a good estimate. Figure 2.2 (c) shows the ratio converges to some constant b greater than 0, therefore the guess ratio test concludes $g(n) = n^{1.1}$ is a good estimate, at least better than $c = 2.0$ or $c = 0.5$ for the growth rate of $t(n)$. However, the guess ratio test cannot conclude $c = 1.1$ is the best estimate, since this experiment tested only a finite number of input sizes, where input size $n \leq 200000$. Figure 2.2 (d) shows the guess ratio curve for $g(n) = nlog(n)$ for $n \leq 200000$. The shape of curve is similar to Figure 2.2 (c), which confirms that $c = 1.1$ is a proper estimate.

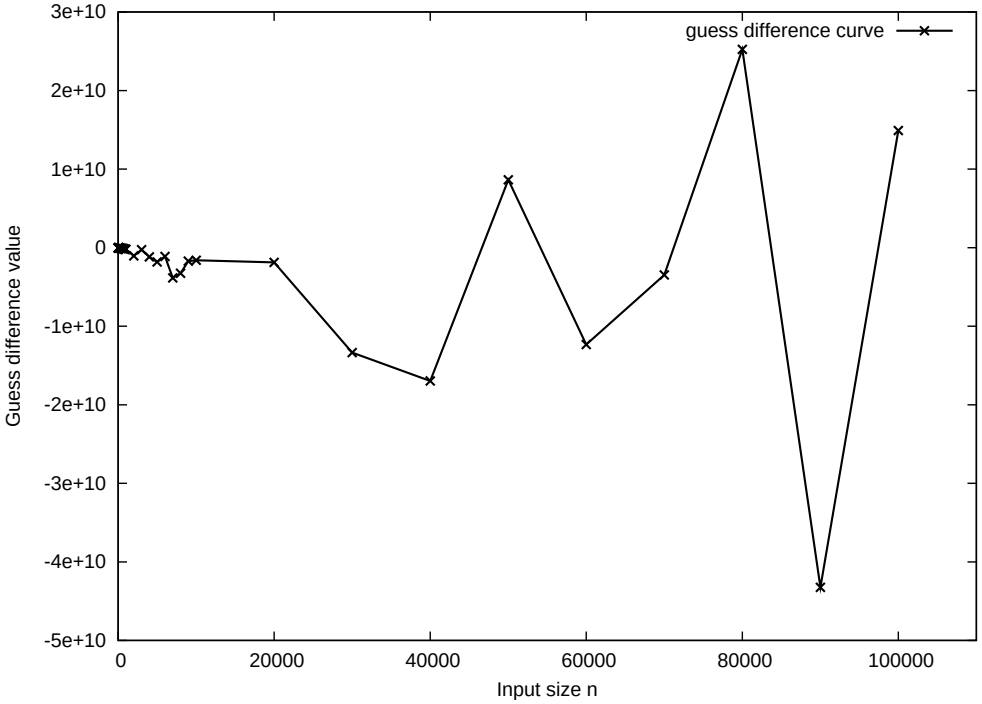
2.4.3 Guess difference test

In the sense of iterating over guess functions, the guess difference test (McGeoch *et al.* , 2002) works similarly to the guess ratio test. The difference is

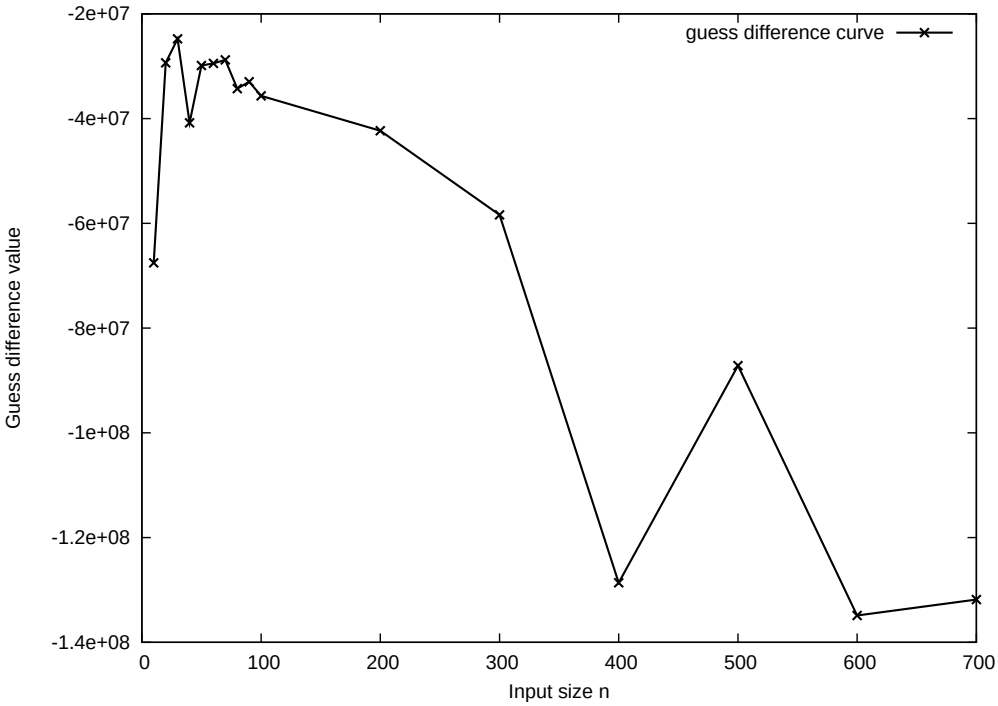
that, rather than evaluating the guess ratio curves, the idea of the guess difference test is to evaluate the difference defined as $g(n) - t(n)$. The test begins with guessing a function having the form $g(n) = an^b$, where a and b are positive rationals. In theory, if the difference curve increases monotonically with n then the guess function $g(n)$ is not $O(t(n))$; if the difference curve monotonically decreases in the range from n_1 to n_k then monotonically increases after n_{k+1}, the guess difference test concludes that the guess function $g(n)$ has the "Down-Up" property (Figure 2.3 gives two examples). Then the test needs to search for other difference curves that have the "Down-Up" property by adjusting the coefficient a until a new "Down-Up" curve is found. When that happens, the guess function is assumed to overestimate the exponent b of $t(n)$. In this case, the guess difference test needs to try another exponent, namely b' where $0 < b' < b$, and applies the same "Down-Up" curve searching procedure again.

After a certain user-specified number of searches, the lowest exponent b for which the test finds a "Down-Up" curve corresponds to the least upper bound of $t(n)$. Note that, if $t(n)$ is a polynomial function of the form $a_1 n^{b_1} + a_2 n^{b_2} + \cdots + a_m n^{b_m}$ where $a_i > 0$ and $b_i > 0, b_i \geq b_{i+1}$ then the guess difference test may find a difference curve having more than one "Down-Up" range. In this case, the test may fail if the input sizes used in the test for the underlying algorithm are not large enough. This is a practical problem that can be seen in Figure 2.3, where the guess difference plot of real experimental data shows the difference curves have more than one "Down-Up" range.

This kind of behavior can be due to the underlying algorithm exhibiting a polynomial time function, but can also be due to $t(n) = t'(n) + E$, where E is random noise. In theory, it is assumed that $t(n_k) < t(n_{k+1})$ holds true for all n. In practice, $t(n_k) \geq t(n_{k+1})$ could be true, in the case where $t'(n_k) + E_{n_k} \geq t'(n_{k+1}) + E_{n_{k+1}}$. In other words, whether the guess difference test can find a reasonable least upper bound for $t(n)$ depends on how the random noise E is dealt with. Whether random noise can be precisely modeled is still an



(a) $n \leq 100000, a = 270, b = 2.0$



(b) $n \leq 700, a = 270, b = 2.0$

Figure 2.3: Guess difference plots of SMO classifier (WEKA implementation of support vector machines) for $g(n) = 270n^2$

open question. Another problem is that the choice of a proper coefficient a for the guess function is not obvious for machine learning algorithms, because the coefficient is related to the number of primitive operations. Except by using special counting methods in the program code, counting primitive operations is not feasible for most machine learning algorithms.

Figures 2.3 (a) and (b) show results obtained by running WEKA's SMO classifier on a data set consisting of three nominal attributes, six numeric attributes and one class attribute. Figure 2.3 (a) shows there is a minimum point at the location where $n = 90000$ in the range $n = 1$ to 100000. The "Down-Up" property that appears between $n = 60000$ and $n = 100000$ is the one that the guess difference test searches for. 2.3 (b) shows there is more than one "Down-Up" range between $n = 1$ and $n = 100$. These "Down-Up" properties should not be counted as being what the guess difference test looks for. They are generated by random noise. It is possible to reduce this noise by using a large offset between each observation. But possible offset values can only be evaluated during experiments.

Guess ratio and guess difference are not stable when the input sizes for the observations are small. Many machine learning algorithms have polynomial running time bounds. The guess ratio test and guess difference test are designed to estimate the exponent of the first term in a polynomial function, and treat other terms as random noise. Therefore, these two tests cannot be guaranteed to find a suitable function when the input size is not large enough to smooth over the random noise. Also, since both tests work by iterating over an unpredictable number of guess functions, the computational cost might be too high for a real time prediction task.

2.4.4 Power test (log-log transformation) and simple linear regression

As in the guess difference test, the power test method (McGeoch *et al.* , 2002; Goodrich & Tamassia, 2002) also assumes $t(n)$ can be formulated by $g(n) = an^b$ where a and b are positive rationals. To find the proper a and b, the power test applies a logarithmic transformation on each $\{n, t\}$ pair in the observations. Secondly, it examines the new $\{n', t'\}$ pairs, where $n' = logn, t' = logt$, to see whether they can be fitted by a simple linear regression line. In what follows, we consider the simple linear regression is based on the least-squares algorithm (Section 2.4.8).

Simple linear regression is a method that studies the relation between a response variable y and a single explanatory variable x. It assumes that for each value of x, the observed values of the response variable y are normally distributed about a mean that depends on x. The statistical model for simple linear regression states that the observed response y_i when the explanatory variable takes the value x_i is $y_i = b_0 + a_1x_i + e_i$, $y_i = b_0 + a_1x_i$ is the mean response when $x = x_i$, and e_i are the deviations that are assumed to be normally distributed with mean 0 and standard deviation s; and e_i is also referred to as the random error.

Using simple linear regression for empirical algorithm analysis, the input size n to an algorithm can be seen as the explanatory variable; the observed running time t of an algorithm can be seen as the response variable. The $\{n, t\}$ pairs in the observations are fitted by the line $t_i = b_0 + a_1n_i$.

Figure 2.4 shows an example of using simple least-squares-based simple linear regression to fit SMO's running time, using the same data as Sections 2.4.2 and 2.4.3. In addition, simple linear regression can be used as an inference model. For example, the simple linear regression line based on observations can be used to make an inference about the running time for a given input size of an algorithm. From Figure 2.4, it can be seen that simple linear regression

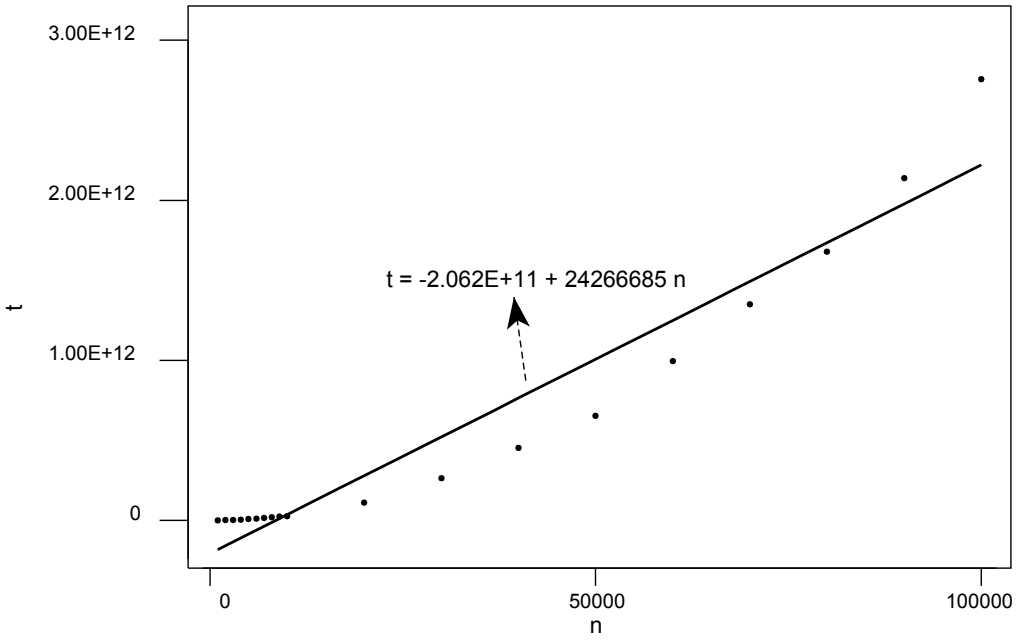


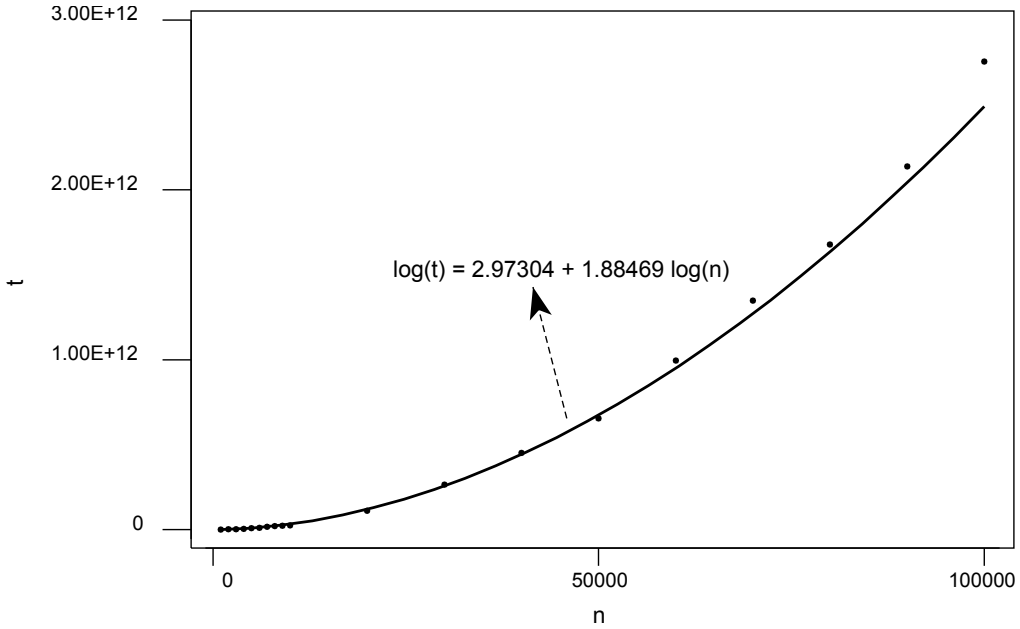
Figure 2.4: Fitting running times of SMO using simple linear regression

gives $t = -2.062E11 + 24266685n$ as the model. To predict the running time of an unobserved input size, such as 200,000, the predicted running time is $t = -2.062E11 + 24266685 \times 200000$.

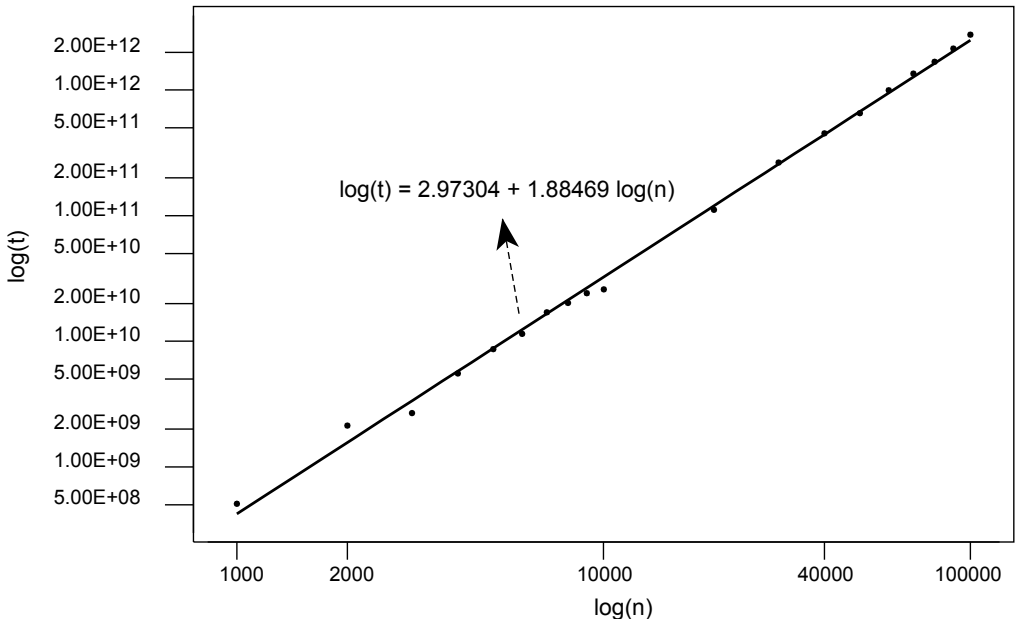
Back to the power rule context. If the transformed observations can be fitted by a linear regression line, such as $t' = b'n' + a'$, then in the power test we can conclude that the proper candidates for a and b of $g(n)$ are $exp(a')$, the intercept, and b', the slope of the fit line, respectively. That is, $t(n)$ can be approximated by $g(n) = exp(a')n^{b'}$.

Figures 2.5 (a) and (b) show two versions of fitted regression lines on the observations for SMO (support vector machines algorithm implemented in WEKA). In (a), the x-axis and y-axis are in the original scale; in (b), the x-axis and y-axis are in the log (log10) transformed scale. The data used consists of three nominal attributes, six numeric attributes and one class attribute. From Figure 2.5 (b), we can see that the transformed data points can be fitted well by a regression line. Therefore, for this experiment, the power test gives $g(n) = exp(2.97307)n^{1.88469}$ as the estimation model, and $O(n^{1.88469})$ as an estimated bound for the time complexity.

There are some other situations. For instance, if the $\{n', t'\}$ pairs grow



(a) The fitted line of a simple linear regression is transformed into logarithmic scale, becoming a curve



(b) Fitted line of a simple linear regression, log-log transformation appied to the data points

Figure 2.5: Power test plot for SMO

in a significant way, but cannot be fitted by a regression line, the power test deduces that $t(n)$ is super-polynomial. In another case, if the $\{n', t'\}$ pairs converge to a constant, then the power law rule concludes that the growth of $t(n)$ is much slower than the polynomial function $g(n) = an^b$, so $t(n)$ can be sub-linear.

In some special cases, when $f(x)$ contains low-order terms, the log-log transformed data points do not lie on a straight line. McGeoch *et al.* (2002) suggested a variation of the power rule, namely that using only the transformed data points at the j highest X values might result in a better asymptotic fit than using all k data points.

2.4.5 Box-Cox transformation

The Box-Cox transformation presented in (Box & Cox, 1964) is a computational method for determining a power transformation for the response variable. The reason for doing such a transformation in the general regression case is that sometimes the response plot y versus explanatory x is curved, and therefore there is a nonlinear relationship between x and y. In this case, if the mean function $E(y|x)$ cannot be summarized by the simple linear regression on x, then a suitable monotonic transformation $T(y)$ of y may turn the nonlinear relationship into a linear one. Table 3.1 shows some common transformations and possible applications.

The Box-Cox method is a numerical procedure for choosing a response transformation $T(y)$ that makes $E(y|x)$ as close to normally distributed as possible. In standardized form, it is defined as

$$T(x) = y^\lambda, y^\lambda = \begin{cases} \dfrac{y^\lambda - 1}{\lambda y^{\lambda-1}} & \text{if } \lambda \neq 0 \\ \bar{y} log(y) & \text{if } \lambda = 0 \end{cases},$$

where $\bar{y}$ is the geometric mean of y.

A more detailed discussion of how to apply Box-Cox transformations for

Transformation	Comments
$\sqrt{y}$ or $\sqrt{y} + \sqrt{y+1}$	Appropriate when $Var(y \mid x) \propto E(y \mid x)$.
$log(y)$	Though most commonly used to achieve linearity, this is a variance stabilizing transformation when $Var(y \mid x) \propto [E(y \mid x)]^2$.
$\frac{1}{y}$	The inverse transformation stabilizes variance when $Var(y \mid x) \propto [E(y \mid x)]^4$.
$sin^{-1}(\sqrt{y})$	This is usually called the arcsine square-root transformation. It stabilizes variance when y is a proportion between zero and one.

Table 2.1: Cook & Weisberg (1999) gives a list of common transformations and possible applications

empirical algorithm analysis can be found in Section 3.4

2.4.6 Ladder transformations

Ladder transformations are of the form $T(y) = y^k$ or $T(x) = x^k$, which belongs to the power family of transformations. It provides a set of transformations for "straightening" a single bend in the relationship between two variables, and is referred to as a family of "one-bend" transformations (Tukey, 1977; McGeoch *et al.* , 2002). These transformations can be used on either x or y. If the transformations are ordered according to the exponent k, a sequence of power transformations is given. In (Mosteller & Tukey, 1977; Tukey, 1977), this is called the transformation ladder. For example:

$$k = -1, -\frac{1}{2}, 0, \frac{1}{2}, 1, 2,$$

where the power transformation $k = 0$ is to be interpreted as the logarithmic transformation.

In applying the idea of ladder transformations to running time prediction, the procedure is to try several transformations of n for the $\{n, t\}$ pairs in the observations, where n corresponds to the explanatory variable x, and t corresponds to the response variable y in the simple linear regression model;

and then to choose that transformation $T_i(n)$ which makes the points most nearly collinear. We can use ladder transformations only on the input size n, and not the running time t, because t is the value to be estimated.

To predict the mean running time for an unobserved input size m, firstly, the input size m is transformed using the transformation $T_i(n)$. Secondly, a simple linear regression model is built based on the transformed observations, and then this linear regression model is used to draw inferences about the mean running time t_m of the explanatory variable $T_i(m)$. The application of ladder transformations for empirical algorithm analysis is discussed, with examples, in Section 3.5.

2.4.7 Curve fitting for extrapolation

In the context of this work, we assume that we do not know the mathematical form for a given algorithm's running time complexity. Our solution is to use the data obtained from the sampling stage; then we analyse the observations to get a deeper understanding of the data. In the modeling stage, we build a model for the observations using interpolation methods, and then extend the model, and extrapolate from there. Here we are interested in the data outside the known data, so we are facing an extrapolation problem, which is the process of constructing new data points outside a discrete set of known data points. It is similar to the process of interpolation. In fact some interpolation methods can also be applied as extrapolation methods. However, the result of using interpolation to solve an extrapolation problem is subject to great uncertainty. Regression is the most commonly employed method that uses both interpolation and extrapolation for practical data analysis and inference. In the following sections, we introduce some regression methods that have been used as the base model for the running time estimators proposed in this work.

Before we start discussing the least-squares problem, which is fundamental to many regression algorithms, we first give a more precise definition for polynomial curve fitting, because the least-squares problem can be seen as a

variation of polynomial curve fitting.

Given observation data points $(x_i, f(x_i)), i = 1, 2, \ldots, N$, a curve fitting function is (Li *et al.* , 2000):

$$P(x) = \sum_{k=0}^{n} a_k \varphi_k(x), n \ll N,$$

where $\varphi_0(x)$, $\varphi_1(x)$, $\cdots$, $\varphi_n(x)$ are the primary functions, that can be any function, such as power functions, trigonometric functions or exponential functions, depending on the actual problem. Let $\varphi_k(x) = x^k, k = 0, 1, \cdots, n$. Then we have the fitting polynomial

$$P_n(x) = a_0 + a_1 x + \cdots + a_n x^n = \sum_{k=0}^{n} a_k x^k.$$

Such a polynomial is usually applied in fitting data points that have no obvious pattern or monotonicity (Li *et al.* , 2000). In predicting the running time of an algorithm, we assume the running time increases as the input size increases. Also, experimental data we obtained for this work confirmed that, for most machine learning algorithms, the running time observation data has a clear pattern of monotone increase. Considering that the running time data points are not noise free and considering this property of monotonicity (for details, please see Figures E.1 to E.6 in Appendix E), we apply and examine several different primary functions, as well as combinations of primary functions, to achieve the goal of finding a proper curve that not only fits the observed data well, but also satisfies the monotonicity assumption for extrapolation.

2.4.8 Least-squares

Least-squares (Lawson & Hanson, 1974; Birkes & Dodge, 1993; Ellis, 1998; Li *et al.* , 2000; Christensen, 2001) is among the most commonly used curve fitting methods. The "best fit" in the least-squares sense is that instance of the model for which the sum of squared residuals has its least value. Most of the

other methods discussed in subsequent sections actually differ from this only by how the "best fit" is defined, but the underlying mathematical descriptions are basically the same. Although they are often discussed in statistical contexts as regression methods, at this stage, we see them as optimization problems since we now are focusing on curve fitting. So, in this section, we place more weight on the theoretical aspects of the least-squares problem from an optimization point of view.

Let r_i be the residual at x_i defined by $r_i = f(x_i) - P_n(x_i)$, where $i = 1, 2, \cdots, N$, $f(x_i)$ is the observed value, and $P_n(x_i)$ is the predicted value, or the approximate value. The least-squares method constructs a curve fitting function having the following form (Li *et al.* , 2000)

$$\Phi(a_0, a_1, \cdots, a_n) = \sum_{i=1}^{N} r_i^2 = \sum_{i=1}^{N} (f(x_i) - \sum_{k=0}^{n} a_k x_i^k)^2. \tag{2.1}$$

In the running time prediction context, suppose we need to weight the importance of each observation (data point) by its value, which corresponds to the input size of an algorithm run instance. For instance, the larger the input size value an observation has, the more important it is. In machine learning or statistics, this weighting is usually implemented as an instance or data point weighting procedure. We mentioned instance weighting because in (Lawson & Hanson, 1974), the authors demonstrate that instance weighting can be employed to smooth the response variable of a simple linear regression model. It may force the underlying regression algorithm to construct a monotonically increasing model, which satisfies the monotonicity assumption for the relation between an algorithm's running time and its input size. To achieve weighting, we can simply add weights to the above function (2.1), so it becomes

$$\Phi(a_0, a_1, \cdots, a_n) = \sum_{i=1}^{N} w_i r_i^2 = \sum_{i=1}^{N} w_i (f(x_i) - \sum_{k=0}^{n} a_k x_i^k)^2.$$

The goal of least-squares curve fitting is to search for a curve so that the square

of the residual r_i for each data point is as small as possible, which is equivalent to the following unconstrained minimization problem (Li *et al.* , 2000)

$$\min_{a_k \in R} \Phi(a_0, a_1, \cdots, a_n) = \min_{a_k \in R} \sum_{i=1}^{N} w_i (f(x_i) - \sum_{k=0}^{n} a_k x_i^k)^2.$$

To find the solution for the problem above, which can be seen as a general optimization problem, we can define the objective function as (Gill *et al.* , 1981; Nocedal & Wright, 1999)

$$f(a) = \frac{1}{2} \sum_{j=1}^{m} r_j^2(a),$$

where a is the parameter (solution) vector, and m is the number of data points. If we assemble the r_j into a residual vector defined by

$$r(a) = (r_1(a), r_2(a), ..., r_m(a))^T,$$

we can rewrite f as

$$f(a) = \frac{1}{2} ||r(a)||_2^2.$$

The derivatives of $f(a)$ can be expressed by using the Jacobian determinant J of r (Nocedal & Wright, 1999).

$$\nabla f(a) = J(a)^T r(a), \tag{2.2}$$

$$\nabla^2 f(a) = J(a)^T J(a) + \sum_{j=1}^{m} r_j(a) \nabla^2 r_j(a). \tag{2.3}$$

We assume the function $r(a)$ is linear, and is of the form $r(a) = Ja + r$, where $r = r(0)$. Therefore, we have (Nocedal & Wright, 1999)

$$f(a) = \frac{1}{2} ||Ja + r||_2^2, \tag{2.4}$$

and we also have

$$\nabla f(a) = J^T(Ja + r), \tag{2.5}$$

$$\nabla^2 f(a) = J^T J. \tag{2.6}$$

Note that the second term of $\nabla^2 f(a)$ defined in (2.3) disappears in (2.6), because $\nabla^2 r_j(a) = 0$ for all j.

In this special case, by setting $\nabla f(a^*) = 0$, and because we know that (2.4) is a convex function, we get the well-known normal equations for (2.4) (Nocedal & Wright, 1999)

$$J^T J a^* = -J^T r. \tag{2.7}$$

Next, we will show a special case of the least-squares problem and its application to the context of running time curve fitting, which is called the linear least-squares problem. In Section 2.4.11, we will consider another special case of the least-squares problem: the non-negative least-squares problem.

2.4.9 Linear least-squares

In Section 2.4.4, we have shown how to use the power rule with simple linear regression to fit running time data, but without a description of the underlying mathematical concepts. Actually, simple linear regression is a "simple" case of multiple linear regression since it has only one explanatory variable. The simple linear regression we used for Section 2.4.4 is based on the least-squares algorithm. Here, we will show an example of the use of the linear least-squares method to fit the running time data of the NaiveBayes classifier (a Naive Bayes implementation in WEKA).

Table 2.2 shows an example of running time observations. The left-hand column contains the input size values, and the right-hand side contains the running time values. Each row in the table can be seen as a running time data point. Therefore, we have seven data points in total. Using these data we would like to obtain an equation that expresses the input size as an ap-

input size n	running time t (ms)
160 (10×2^4)	1.0
320 (10×2^5)	1.0
640 (10×2^6)	4.0
1280 (10×2^7)	8.0
2560 (10×2^8)	21.0
5120 (10×2^9)	39.0
10240 (10×2^{10})	102.0

Table 2.2: Running time data of WEKA's NaiveBayes classifier building models on different sizes of input of an artificial data set

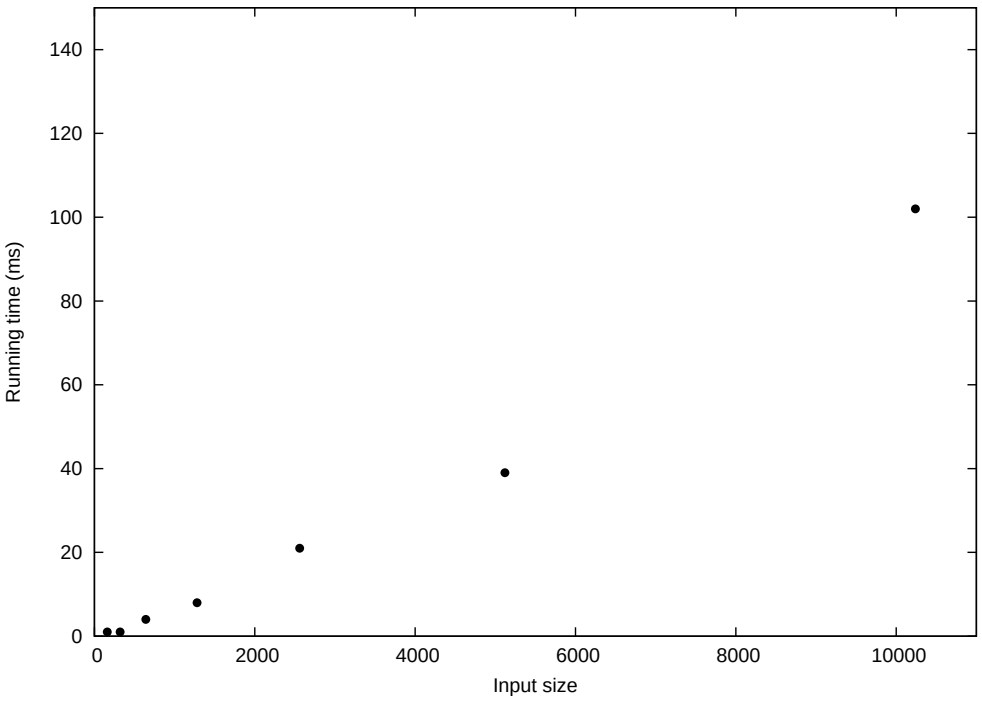


Figure 2.6: Plot of the running time data in Table 2.2

proximate function of the running time measurement. For notation, let y_i be the running time measurement of the ith data point, and x_i be the input size of the ith data point. The data points are plotted in Figure 2.6. We use the following linear model

$$Y = a_0 + a_1 X_1 + a_2 X_2 + \cdots + a_p X_p + e, \tag{2.8}$$

where e is the random error with a mean of 0. We are interested in constructing an approximate equation for the data. For this example, we set the value of p to be 2, where $X_1 = n$, and $X_2 = n^{1.1}$. In Chapter 3, we shall show that theoretically p can be any integer greater than 0. Here we can re-write (2.8) as

$$Y = a_0 + a_1 n + a_2 n^{1.1} + e.$$

In terms of the observed data the model is

$$y_i = a_0 + a_1 n_{i1} + a_2 n_{i2}^{1.1}, \tag{2.9}$$

for $i = 1, 2, ...7$. The least-squares estimates of a_0, a_1 and a_2 are defined as the $\hat{a}_0, \hat{a}_1$ and $\hat{a}_2$ that give the least sum of squares of the residuals $\sum r_i^2$, where $r_i = y_i - (\hat{a}_0 + \hat{a}_1 n_{i1} + \hat{a}_2 n_{i2})$. We can see that this dovetails with the optimization problem expressed by (2.4). In matrix notation, model (2.9) can be expressed as

$$y = Xa + e. \tag{2.10}$$

The normal equations for (2.10) are (Birkes & Dodge, 1993)

$$(X^T X)\hat{a} = X^T y,$$

and the algebraic solution of the normal equations can be written as

$$\hat{a} = (X^T X)^{-1} X^T y. \tag{2.11}$$

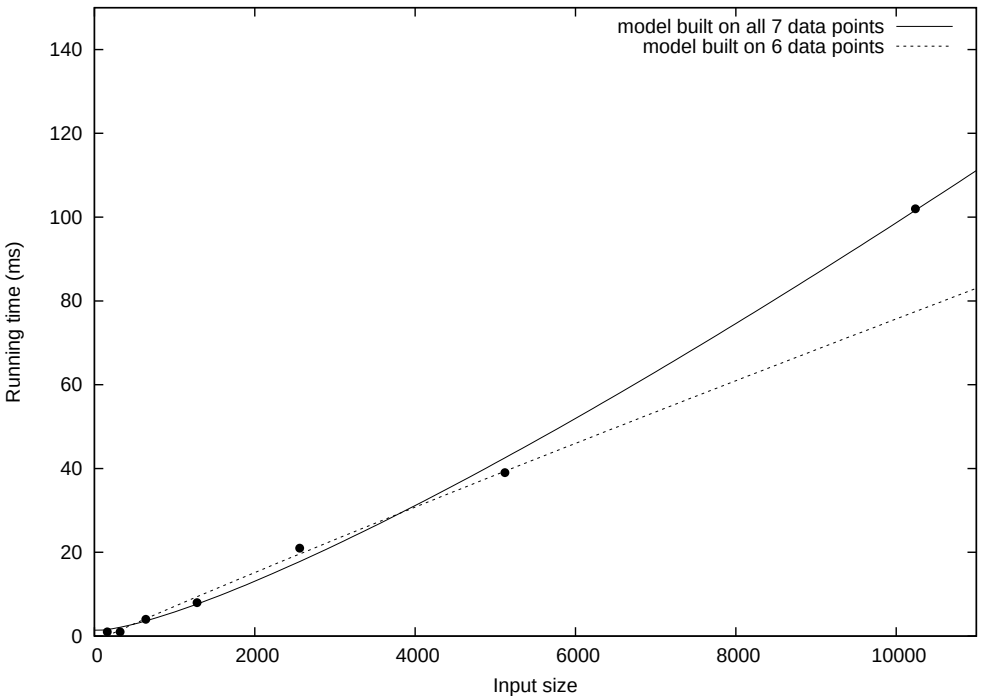


Figure 2.7: Plot of the running time data in Table 2.2 fitted by two models

For the data in Table 2.2, Formula (2.11) yields the following estimates

$$\begin{bmatrix} \hat{a_0} \\ \hat{a}_1 \\ \hat{a}_2 \end{bmatrix} = \begin{bmatrix} 1.41 \\ -0.0164 \\ 0.0104 \end{bmatrix}.$$

Therefore the estimated regression equation is $\hat{Y}_{all} = 1.41 - 0.0164X_1 + 0.0104X_2$. If we use the first six data points in Table 2.2 and not the last one ($n = 10240$), the estimated regression equation is $\hat{Y_6} = -1.37 + 0.115X_1 - 0.00151X_2$.

Figure 2.7 shows the fitted data. The solid curve $\hat{Y}_{all}$ is built on all seven data points, the dashed curve $\hat{Y_6}$ shows the least-squares linear regression model built upon the first six data points. We can see that the shapes and the model parameters are quite different. This is due to the nature of curve fitting methods, the goal of which is to find a close fit to the observed data, and the focus is not on the extrapolative capability regarding unknown data.

2.4.10 Least absolute deviations

In the method of least-squares, the parameters for the linear regression model are chosen so that the sum of the squares of the residuals, $\sum r_i^2$, is as small as possible. In the method of least absolute deviations (LAD), the parameters are chosen so that the sum of the absolute values of the residuals, $\sum |r_i|$, is as small as possible. That is, LAD estimates the model parameters that minimize $\sum |y_i - Xa|$. The concept of LAD is similar to the concept of least-squares estimation. However, in the actual calculation of the parameter estimates, the LAD method is more complicated since there are no formulas for the LAD estimates (Birkes & Dodge, 1993).

Birkes & Dodge (1993) present a numerical method. The procedure is that, for any given data point (x_i, y_i), we find the best line among all the lines passing through it on (x_0, y_0). That is, for each data point (x_i, y_i) calculate the slope $\frac{y_i - y_0}{x_i - x_0}$ of the line passing through the two points (x_0, y_0) and (x_i, y_i). In the case that $x_i = x_0$, the slope is not defined so that point is ignored. Then index the data points so that (Birkes & Dodge, 1993)

$$\frac{(y_1 - y_0)}{(x_1 - x_0)} \leq \frac{(y_2 - y_0)}{(x_2 - x_0)} \leq \cdots \leq \frac{(y_n - y_0)}{(x_n - x_0)}.$$

Let $T = \sum |x_i - x_0|$. The least absolute deviation problem is equivalent to the problem of finding the index k that satisfies the following conditions (Birkes & Dodge, 1993)

$$|x_1 - x_0| + \cdots + |x_{k-1} - x_0| < \frac{1}{2}T, |x_1 - x_0| + \cdots + |k_{k-1} - x_0| + |x_k - x_0| > \frac{1}{2}T.$$

The best line passing through (x_0, y_0) is the line $Y = b + aX$, for which

$$a = \frac{y_k - y_0}{x_k - x_0}, b = y_0 - ax_0.$$

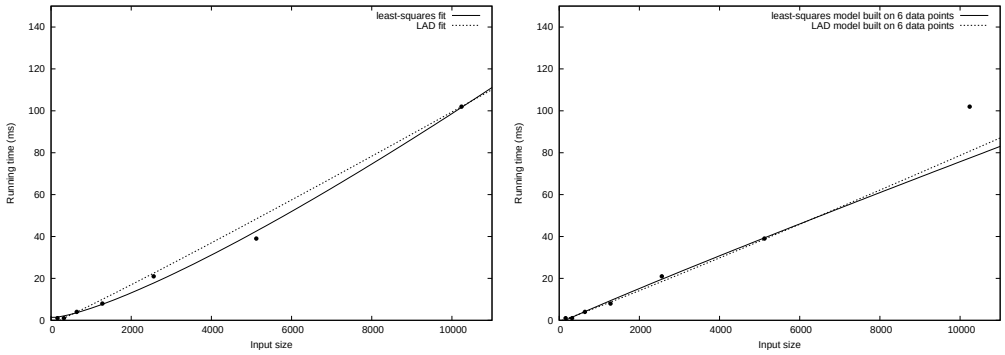
Algorithm 1 An easy to implement algorithm for computing the LAD regression line of a given data set

```
LAD-Lines = [] // a collection

For each data point d in all data points

  line = build_line(d, d') // d' is another data point other than d
  LAD-Lines.add(line)

End For

LAD-Line = get_line_with_min_sum of absolute_deviations(LAD-Lines)
```

Based on the above numerical procedure, Birkes & Dodge (1993) suggested a simple implementation of the LAD algorithm, which has the advantage of being conceptually simpler. Although the simple algorithm has the disadvantage of requiring a greater computation cost than the numerical method, it is still feasible for the running time prediction task considered here. The simple algorithm is based on the observation that an LAD regression line should pass through at least two data points. So an LAD regression line can be found among the lines defined by all possible pairs of data points. Therefore we can simply compute the sum of absolute deviations for each of these lines and choose the one with the smallest sum.

Algorithm 1 shows the simple algorithm's pseudo-code for calculating the LAD regression line. Figure 2.8 (a) shows the curve obtained by applying LAD on the data in Table 2.2, compared with the least-squares fit. Figure 2.8 (b) shows the least-squares and LAD built models on the first 6 data points in Table 2.2. We can see that the two fitted curves are quite close to each other on this data set.

In Section 3.9 and Section 3.10, we will show how to extend the LAD curve fitting method so that it has the capacity to predict the running time of an algorithm.



(a) Models built on all 7 data points (b) Models built on 6 data points

Figure 2.8: Plot of the running time data in Table 2.2 fitted by the least-squares and LAD

2.4.11 Non-negative least-squares

In some situations the least-squares method (or LAD) must be reformulated by the introduction of certain inequality constraints, which may constitute additional information about a problem. For instance, given the running time data in Table 2.3, we apply the least-squares method to fit the data by using the following linear model

$$Y = a_0 + a_1X_1 + a_2X_2 + a_3X_3 + a_4X_4 + e,$$

where $X_1 = n$, $X_2 = log(n)$, $X_3 = n^{1.1}$ and $X_4 = n^3$. Then, we get the fitted least-squares curve $Y = 35.45 + 0.0018X_1 - 9.9824X_2 - 0.0005X_3 + 0.0X_4$. Figure 2.9 shows the curve. We can see that the least-squares model in this case is not reasonable because we want the predictions to be non-negative. Therefore, we need to restrict the method to return non-negative predictions.

The idea is that if we use a linear polynomial for the regression model, with non-negative coefficients, the model should be monotonic or convex, ideally giving increasing predictions while the input size n increases, which satisfies our assumption that the running time of an algorithm increases when the input size increases. Such a special case of linear least-squares with linear inequality constraints (LSI) is usually called the non-negative least-squares (NNLS) problem. The LSI problem is an optimization problem, that is defined

Input size n	Running time t (seconds)
20000	4.61
30000	7.47
40000	10.19
50000	13.37
60000	16.00
70000	18.79
80000	22.22
90000	24.67
100000	27.93

Table 2.3: Running time data

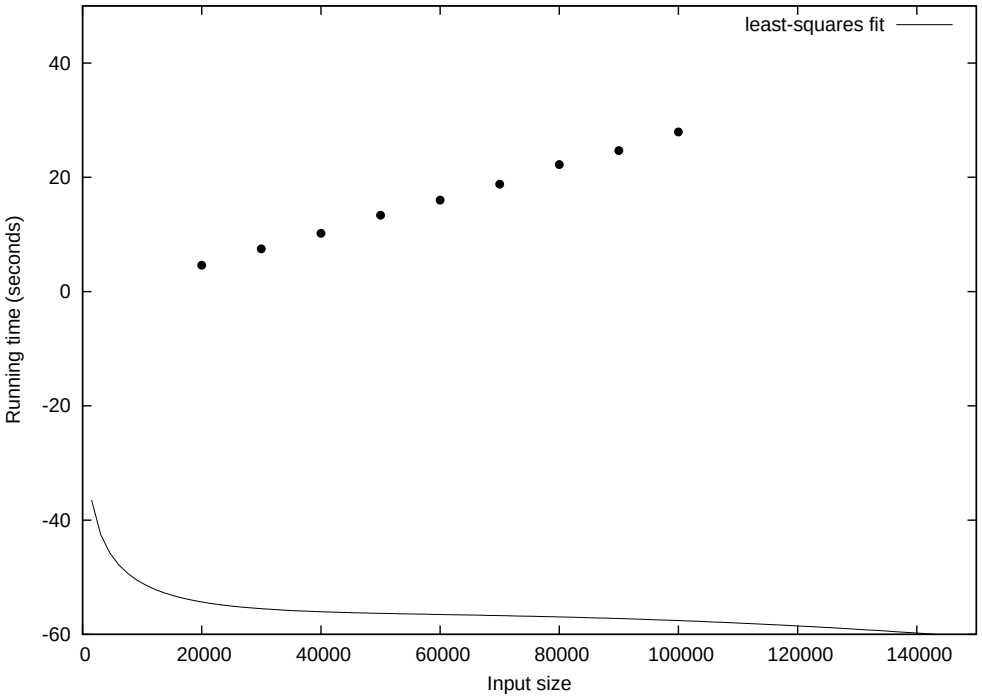


Figure 2.9: Plot of the running time data in Table 2.3 fitted by least-squares

as (Lawson & Hanson, 1974)

$$min||Xa - f|| \text{ subject to } Ga \geq h.$$

The NNLS problem is a restricted form of the LSI problem, and can be written as

$$min||Xa - f|| \text{ subject to } a \geq 0.$$

LSI problems can be solved using standard numerical methods, such as line search and trust region methods, in particular the very popular BFGS

algorithm, named for its discoverers Broyden, Fletvher, Goldfard, and Shanno (Gill *et al.* , 1981; Nocedal & Wright, 1999). However, when investigating the BFGS method, our experimental results showed that it is not suitable for the sampling-based running time prediction problem with small sample size. In particular, when the number of explanatory (independent) variables is greater than the number of data points, the BFGS method may fail to find a solution (Nocedal & Wright, 1999). Therefore, for this work, we employed an algorithm found in the literature on non-standard constrained optimization. As discussed in (Lawson & Hanson, 1974), the conditions characterizing a solution for LSI are subject of the Kuhn-Tucker theorem:

An n-vector $\hat{a}$ is a solution for LSI if and only if there exists an m-vector $\hat{y}$ and a partitioning of the integers 1 through m into subsets ε and ξ such that

$$G^T\hat{y} = X^T(X\hat{a} - f),$$

$$\hat{r}_{i\in\epsilon} = 0, \hat{r}_{i\in\xi} > 0,$$

$$\hat{y}_{i\in\epsilon} \geq 0, \hat{y}_{i\in\xi} = 0,$$

where $\hat{r} = G\hat{a} - h$.

Further discussion of this theorem, including its proof, can be found in (Fiacco & McCormick, 1968).

Based on this theorem, Lawson & Hanson (1974) gave a finite convergence algorithm to solve the problem NNLS. Pseudo-code for this algorithm is listed in **Algorithm 2**. On termination the solution vector a satisfies

$$a_j > 0, j \in P;$$

$$a_j = 0, j \in Z,$$

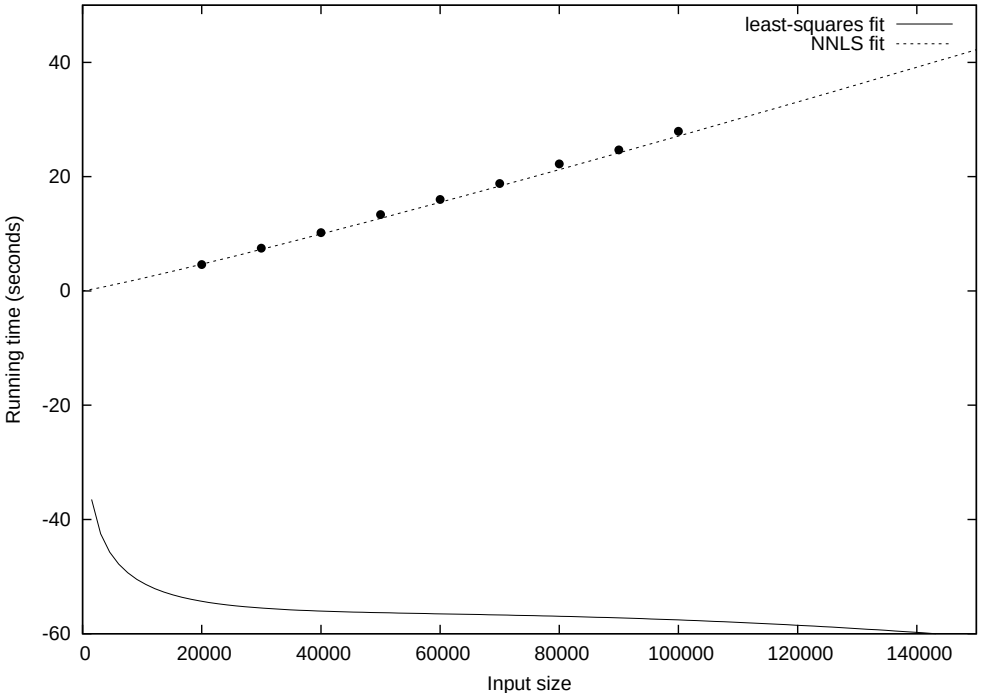


Figure 2.10: Plot of the running time data in Table 2.3 fitted by two models

Algorithm 2 NNLS(X, m, n, f) solving the non-negative least-squares problem, adapted from (Lawson & Hanson, 1974)

Step	Description
1	Set $P := NULL$, $Z := 1, 2, ..., n$, and $x := 0$.
2	Compute the n-vector $w := X^T(f - Xa)$.
3	If the set Z is empty or if $w_j \leq 0$ for all $j \in Z$, go to Step 12.
4	Find an index $t \in Z$ such that $w_t = max\{w_j : j \in Z\}$.
5	Move the index t from set Z to set P.
6	Let X_P denote the $m \times n$ matrix defined by Column j of $X_P := \begin{cases} column & j \text{ of } X \text{ if } j \in P \\ 0 & \text{if } j \in Z \end{cases}$. Compute the n-vector z as a solution of the least-squares problem $X_P z \cong f$. Note that only the components $z_j, j \in P$, are determined by this problem. Define $z_j := 0$ for $j \in Z$.
7	If $z_j > 0$ for all $j \in P$, set $a := z$ and go to Step 2.
8	Find an index $q \in P$ such that $x_q/(x_q - z_q) = min\{x_j/(x_j - z_j) : z_j \leq 0, j \in P\}$.
9	Set $\alpha := a_q/(a_q - z_q)$.
10	Set $a := a + \alpha(z - a)$.
11	Move from set P to set Z all indices $j \in P$ for which $y_j = 0$. Go to Step 6.
12	The computation is completed and output a.

and is a solution vector for the least-squares problem

$$X_P a \cong f.$$

The proof of the convergence of this NNLS algorithm is discussed in Appendix A.

Next, we apply the NNLS algorithm to fit the data in Table 2.3. Figure 2.10 shows the fitted NNLS curve $Y = 0.000018X - 0.0X_2 - 0.00008X_3 + 0.0X_4$, where $X_1 = n$, $X_2 = log(n)$, $X_3 = n^{1.1}$ and $X_4 = n^3$. We can see that the NNLS curve fits the data very well, and is much more reasonable than the unconstrained least-squares fit based on the same linear regression model.

2.5 Conclusions

We have considered the case where $f(n)$ is the running time function of an algorithm which we want to estimate, where n is the input size. The goal of this work is to find a method that can automatically construct an estimation model $g(n)$ based on sampling techniques, and then use this model to predict the running time t where $t = f(n)$. In this chapter, we discussed several curve fitting methods. These methods are fundamental to the running time estimators proposed in this work. Some of them can be applied directly to an extrapolation problem, some of them need to be adapted. The goal is to find methods that can construct a model that not only maps the observed data well, but also has the ability to predict unknown data points. In the next chapter, we will describe how to use these curve fitting methods to make predictions—by constructing running time estimators.

Chapter 3

Running time estimators

In the previous chapter, we have discussed curve fitting methods that can potentially be employed as the basis for designing a running time estimator. In this chapter, we will show how to use those curve fitting methods in linear regression models designed for the running time prediction problem. In total there are 11 running time estimators that are considered.

The prediction performance of each estimator is evaluated on running time data obtained from monitoring the running time of 41 WEKA classifiers. The evaluation results are discussed in Chapter 5. Before discussing the estimators, we will explain why we chose to use linear regression as the basic approach, what kind of linear models we have used, and the foundational assumptions made for this work.

In computer science, the Random Access Machine (RAM) (Elgot & Robinson, 1964) is an approach in which counting primitive operations gives rise to a computational model. In the RAM model, we assume an algorithm's time complexity can be expressed by a function of input size, of the form

$$t = Tr(n) + \varepsilon,$$

$$Tr(n) \geq 0, n \geq 0,$$

where t is the running time of the algorithm, n is the input size, Tr is the trend or time complexity function, and ε is the error term. This model says that the running time t can be represented in terms of the input size n according to the equation $g(n) = Tr(n)$ and by the error term ε (see Section 1.1 for a

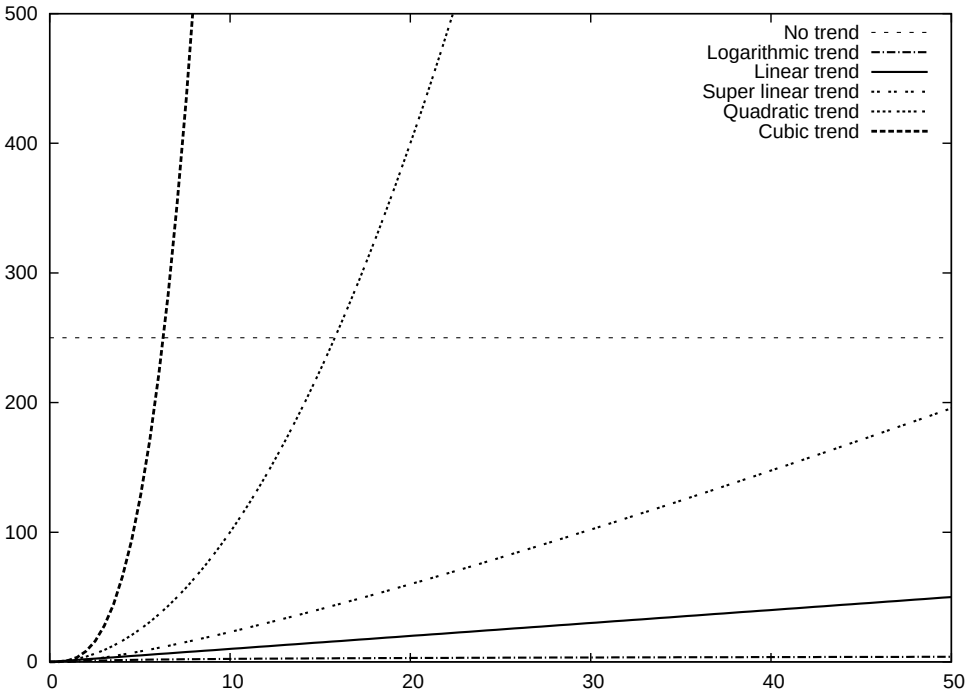


Figure 3.1: Shapes of different running time trends

definition of $g(n)$). This error term represents random factors that cause the running time t to deviate from the average level g. Figure 3.1 shows some useful trends.

Based on a preliminary investigation of experimental data obtained from the running times of 41 WEKA machine learning algorithms, we assume an algorithm's running time complexity is in one of the following categories:

- No trend, which is modeled as $Tr(n) = \beta_0$; this means there is no growth or decline.

- Linear trend, which is modeled as $Tr(n) = \beta_0 + \beta_1 n$; this implies that there is a straight line growth or decline (depending on the value of β_1).

- Logarithmic or square root trend, where there are variations of linear trend, such as $Tr(n) = \beta_0 + \beta_1 k(n)$, where $k(n) = log(n)$ or $k(n) = sqrt(n)$.

- Quadratic trend, which is modeled as $Tr(n) = \beta_0 + \beta_1 n + \beta_2 n^2$; this implies that there is a quadratic change as the input size n grows.

- Super linear and sub-quadratic trends; these are variations of quadratic trend, such as $TR(n) = \beta_0 + \beta_1 j(n) + \beta_2 k(n)$, where $k(n) = nlog(n)$ or $k(n) = n^m, 1 < m \leq 2$, and $j(n)$ is a linear trend function.

- Cubic trend, which is modeled as $Tr(n) = \beta_0 + \beta_1 n + \beta_2 n^2 + \beta_3 n^3$; this implies that there is a cubic change as the input size n grows. As with the linear and quadratic trends, there are variations of cubic trend consisting of linear and quadratic trend functions.

Although there are more complicated trends, in this work, we assume most WEKA algorithms belong to the trends above, and exhibit growth in running time as n increases. We propose the following "full" model for expressing the running time function:

$$t = Tr(n) + \varepsilon = \beta_0 + \beta_1 log(n) + \beta_2 nlog(n) + \beta_3 n^{0.1} + \beta_4 n^{0.2} + \cdots +$$

$$\beta_{12} n^{1.0} + \beta_{13} n^{1.1} + \cdots + \beta_{22} n^{2.0} + \beta_{23} n^{2.1} + \cdots + \beta_{27} n^{2.5} + \varepsilon. \tag{3.1}$$

We then rewrite the above trend model into the form of a linear regression model

$$y = \mu_{y|x_1,x_2,\ldots,x_k} + \epsilon = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + \varepsilon, \tag{3.2}$$

where $\mu_{y|x_1,x_2,\ldots,x_k} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k$ is the mean of the response variable y given the explanatory variables $x_1, x_2, \ldots, x_k$, $\beta_0, \beta_1, \ldots, \beta_k$ are regression parameters (coefficients) to be estimated, and ε is an error term. The error term describes the effects on y of all factors other than the explanatory variables. The number of explanatory variables is $k = 27$ in the above model. We can see that k can be any integer greater than 0, because in theory we can add an unlimited number of trend functions into (3.1).

When we use the linear regression model stated by (3.2), we face a challenging problem: in the case that we have p data points, if $p < 27$, then the number of explanatory variables is greater than the number of data points. This may cause numerical problems when using the curve fitting algorithms discussed

in the last chapter, especially the least-squares, least absolute deviations and the non-negative least-squares algorithms, because standard implementations of those algorithms can not be guaranteed to find a solution vector when the number of explanatory variables is greater than the number of data points. Another problem is that the value of n^3 in a cubic trend model can be very large, and the observed running time value may be too small compared with the value of n^3. This may result in an ill-conditioned matrix that is difficult for numerical methods. In the case that the underlying system is an ill-conditioned matrix, there are methods that can be used to solve this problem (Li *et al.* , 2000). One is to progressively update the values of the explanatory variables of the system until its condition number is acceptable. Another method is to find out the length of significant digits that is required for the solution of the system of equations and make sure the returned regression coefficients are at least in that length. However, neither of the methods can be guaranteed to return reasonable solution vectors for all the running time prediction problems examined in this work.

Thus, to avoid such problems, we reduce the full trend model to a more compact one. Here is an example with only three terms from the full model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2,$$

$$X_1 = n, X_2 = nlog(n).$$

Subsequent sections will provide detailed descriptions of how the estimators are constructed, and what kind of reduced trend models are used.

3.1 Data abstraction

All estimators were programmed in Java, using WEKA as a class library. We here introduce some additional classes that have been created for this work:

- An Observation object is defined as a data structure consisting of an

Observation ID	n	T				
		Run 1 t_1	Run 2 t_2	Run 3 t_3	Run 4 t_4	Run 5 t_5
Observation 1	100					
Observation 2	200					
Observation 3	300					
Observation 4	400					
Observation 5	500					
Observation 6	600					
Observation 7	700					
Observation 8	800					
Observation 9	900					

Table 3.1: A sample Observations object that contains nine Observation objects

$\{n, T\}$ pair, where n is the input size, and $T < t_1, \ldots, t_m >$, which is a vector of running times observed from m runs, for input size n.

- An Observations object is defined as a data structure that is a collection of Observation objects. Table 3.1 shows an example Observations object (without actual observed values).

- An Estimator takes an Observations object as input to build an estimation model that can predict the running time of a given input size.

- The "training observations" object corresponds to the Observations object that is used to build an estimation model.

- The "testing observations" object corresponds to the Observations object that is used to evaluate the prediction performance of a running time estimator.

3.2 Predicting the running time of a machine learning algorithm

A machine learning algorithm learns a target concept from examples. The examples are also called a "training set". Assuming the running time of an

algorithm depends mainly on the size of the input to the algorithm, the running time of a given machine learning algorithm training on a particular training set relates mainly to the size of the training set. Although many other factors may contribute to the running time of an algorithm, these factors are treated as random noise in what follows. Given a machine learning algorithm M and a training set S with k instances, a sampling-based running time prediction method, an "estimator" for short, works as follows:

- Firstly, an Observations object is obtained by observing the running times of algorithm M training on different subsets of S. For example, the size of the training set may be 10,000, that is, a machine learning algorithm is supposed to learn a target concept (build a concept model) based on 10,000 examples. In this case, the information stored in the Observations object is similar to Table 3.1, which is obtained by observing the running time of the machine learning algorithm training on subsets of the 10,000 examples, for instance, $subset_1 = 100, subset_2 = 200, \ldots, subset_9 = 900$.

- Secondly, an estimator builds an estimation model based on the Observations object. Then, the estimator can be used to predict the running time of algorithm M training on the full training set S.

3.3 PSLR—Power rule with simple linear regression

The estimator PSLR (based on the power test discussed in Section 2.4.4) first applies a log-log transformation on each Observation in the Observations object. Secondly, PSLR builds a simple linear regression model based on the transformed Observations object, to predict the running time t for a given input size n. PSLR uses the simple linear regression model to predict response variable t' given explanatory variable n' where $n' = log(n)$. Finally, PSLR

gives the predicted running time as $t = exp(t')$.

3.4 BC—Box-Cox transformations

The estimator BC (based on the Box-Cox method discussed in Section 2.4.5) works by searching for the best simple linear regression model after applying different transformations to the explanatory variable n—the input size. The Box-Cox transformations are controlled by manipulating the exponent λ in the following equation:

$$T(x) = y^{\lambda}, y^{\lambda} = \begin{cases} \frac{y^{\lambda} - 1}{\lambda y^{\lambda - 1}} & \text{if } \lambda \neq 0 \\ \bar{y} log(y) & \text{if } \lambda = 0 \end{cases},$$

where $\bar{y}$ is the geometric mean of y, which is set to be 1 here. In this work, BC applies transformations on n from the range $\lambda = 0$ to $\lambda = 2.5$, where λ is incremented by 0.1 in each step. There are a total of 26 transformations ($\lambda = 0, \lambda = 0.1, \lambda = 0.2, \ldots, \lambda = 2.5$). Thus, BC builds 26 simple linear regression models corresponding to the 26 transformations. The "best" simple linear regression model over these transformations is defined as the one that results in the lowest squared error. To predict the running time t for a given input size n, BC uses the "best" simple linear regression model to predict the response variable t based on the explanatory variable n' where $n' = T(n^{\lambda_{best}})$.

3.5 LADDER—Ladder transformations with simple linear regression

Like estimator BC (Box-Cox estimator), estimator LADDER also uses the "best" simple linear regression model built on transformed explanatory variables. To predict the running time t for a given input size n, the original ladder transformation technique is extended to use a set of ladder transforma-

tions that are designed for predicting an algorithm's running time. That is, simple running time trend functions can be ordered by growth rate (this can be seen as a variation of the full model (3.1)):

$$T(n) = logn, log^2 n, n, nlogn, n^{1.1}, n^{1.2}, n^{1.3}, \ldots, n^{1.9}, n^{2.0}, n^{2.1}, \ldots, n^{2.5}.$$

As with PLSR and BC, LADDER builds different simple linear regression models based on the above ladder transformations, and uses the one that gives the lowest squared error as the estimation model to predict the response variable t based on the explanatory variable n', where $n' = T_{best}(n)$.

3.6 LsF—Least-squares regression on the full trend model

The estimator LsF uses the full running time trend model stated in (3.1). As a consequence the associated linear regression model (3.2) will have 27 explanatory variables. At the beginning of this chapter, we have discussed that if the number of explanatory variables is greater than the number of data points, then the least-squares computation and the resulting coefficient vector may not be trustworthy. In order to reduce the number of explanatory variables in the linear model, M5 feature selection (Wang, 2000; Witten & Frank, 2005), which is the default feature subset selection method for WEKA's linear regression algorithm, is employed. The idea of M5 feature selection is to step through the explanatory variables, removing the one with the smallest standardised coefficient until no improvement is observed in the estimate of the error given by the Akaike information criterion (Akaike, 1974; Hall, 1999; Wang, 2000; Witten & Frank, 2005). Then we will have a reduced feature (explanatory variable) set. Finally, LsF uses the reduced feature set and applies the least-squares algorithm on the training data points.

3.7 LsR—Linear regression on a restricted trend model

Rather than using an automated feature selection method to reduce the size of the explanatory variables, the estimator LsR uses a restricted linear regression model consisting of three terms, a constant term, a term using X, and a higher-order term, such as $XlogX$, X^2, or X^3. For this work, based on empirical observations, we chose $XlogX$ as the higher-order term, so the restricted model is

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2,$$

$$X_1 = n, X_2 = nlogn, \tag{3.3}$$

where n is the input size of an Observation. The restricted model has the advantage that the estimator requires less time to build its prediction model compared with estimators using a feature subset selection method. Also, the dimensionality of the problem is reduced, so the resulting model is compact and very readable.

3.8 LsSeq—Linear regression using an adapted wrapper method for feature subset selection

The construction of estimator LsSeq is similar to that for the estimator LsF, except LsSeq uses a version of the wrapper method (Kohavi & Sommerfield, 1995; Kohavi & John, 1996) for feature subset selection. Figure (3.2) illustrates the basic structure of a wrapper method. The idea is simple: the induction algorithm is considered as a black box, and the feature selection algorithm conducts a search for a good subset using the induction algorithm itself as part of the function evaluating feature subsets. In a wrapper method, the training data are usually partitioned into internal training and testing sets. The feature subset with the "best" evaluation score on the test data is chosen

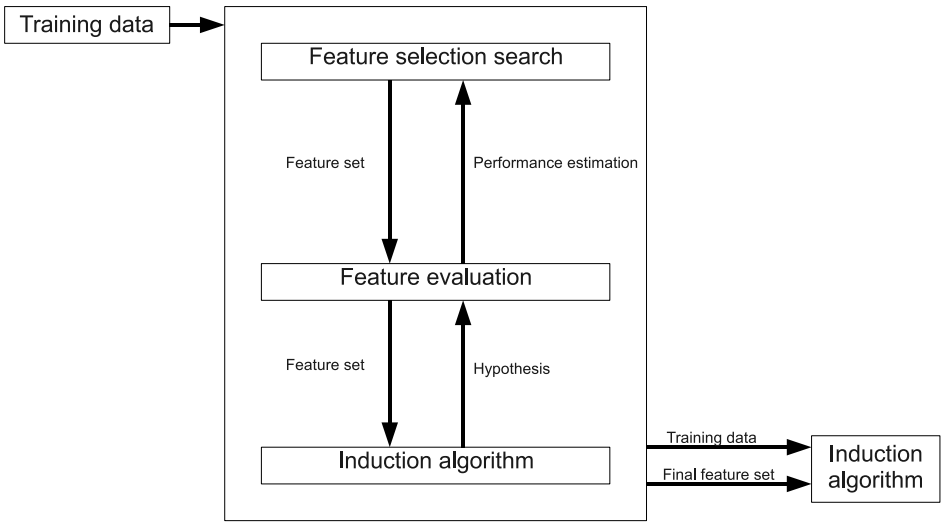


Figure 3.2: The wrapper approach for feature subset selection

Algorithm 3 A version of the wrapper approach to feature subset selection, used by estimator LsSeq

For all numbers 0 < M < N of samples (training data points)

  For all feature subsets

    Build model from first M of N samples using current attribute subset
    Evaluate error of model on remaining $N - M$ samples
    Add observed error to overall error of this feature subset

  End For

End For

Build final model using feature subset with lowest overall error

as the final feature set, which is then used to apply the induction algorithm.

Inspired by the wrapper method, we propose the feature subset selection algorithm shown in **Algorithm 3** for running time estimation. N is the total number of data points available for building the model, corresponding to the number of subsamples of different size used to estimate running time. This procedure automatically identifies the most relevant terms for the regression model based on evaluations in the training stage. It measures the error of the extrapolation for each subset considered, based on using each subsequent training data as training observations and evaluating on the remaining data. Here, we use the absolute difference between the observed value and the predicted value as the evaluation basis.

In preliminary research for this work, we found this method works better than standard attribute selection or regularization because it measures the effect on extrapolation directly. Also, it considers the performance for each training sub-sequence and provides additional useful information: if a set of features is appropriate for extrapolating from a particular training sub-sequence it should also be suitable for extrapolation from all other sub-sequences. In this work, we use up to $N-1$ data points for training, and the Nth data point for testing. The feature subset that results in the smallest absolute difference is used as the final feature set.

3.9 LadF—LAD regression on the full trend model

The estimator LadF works in exactly the same way as estimator LsF, except that LadF uses the least absolute deviations algorithm (Section 2.4.10) as the underlying curving fitting algorithm for the regression model.

3.10 LadR—LAD regression on a restricted trend model

The estimator LadR works in exactly the same way as estimator LsR, except that LadR uses the least absolute deviations algorithm (Section 2.4.10) as the underlying curving fitting algorithm for the regression model.

3.11 nnlsF—NNLS on the full trend model

The estimator nnlsF works in exactly the same way as estimator LsF, except that nnlsF uses the non-negative least-squares algorithm (Section 2.4.11) as the underlying curving fitting algorithm for the regression model.

3.12 nnlsR—NNLS on a restricted trend model

The estimator nnlsR works in exactly the same way as estimator LsR, except that nnlsR uses the non-negative least-squares algorithm (Section 2.4.11) as the underlying curving fitting algorithm for the regression model.

3.13 OneTest—A regression meta learner for running time prediction

In terms of machine learning, a meta learner (Vilalta & Drissi, 2002) studies how to choose the right bias (base learner) dynamically, as opposed to individual base learners where the bias is fixed a priori, or user parameterized. One advantage of a meta learner is that it overcomes the problem that a base learner may perform very well on one problem, but very badly on the next.

In this work, we propose a cross-validation selection based regression meta learner for the running time prediction problem. In our context, a cross-validation selection based regression meta learner is similar to the idea of the

Algorithm 4 A cross-validation selection based regression meta learner for running time prediction

For each estimator e in E

  Do k times (where k is a constant, we suggest using $k = 1$)

    Divide the N training data points into two data sets: $N1$ and $N2$
    (where $N1$ has $N - k$ data points, and $N2$ has k data points)

    Build estimator e with N1
    Test estimator e with N2, and record the evaluation score

  End Do

End For

Select the estimator that obtains the lowest average evaluation score
(the evaluation score is defined as the absolute difference between the observed value and the predicted value)

wrapper method. **Algorithm 4** shows the pseudo-code of the meta learner. We call it OneTest, since when choosing the "best" estimator (base learner) to use, OneTest gives $N - 1$ data points to each estimator for training, and only one data point for testing. The reason for using only one data point for evaluation is that running time prediction is a small sample size problem, and we usually do not have many training data points. For this work, the meta learner OneTest was applied in conjunction with the following ten estimators: PSLR, BC, LADDER, LsF, LsR, LsSeq, LadF, LadR, nnlsF and nnlsR.

3.14 Conclusions

In this chapter, we described how to use the curve fitting methods discussed in Chapter 2 in linear regression models to construct running time estimators. At the beginning, we explained that why the full trend model stated in Equation 3.1 needs to be reduced to a more compact one. We described with examples that how to adapt the least-squares-, LAD- and NNLS-based algorithms for running time prediction. Also, the feature selection methods employed by the

running time estimators are discussed in detail. We also proposed a regression meta leaner called "OneTest" that can be applied in conjunction with any individual estimators. In Chapter 5, we will show the experimental results using different evaluation methods.

Chapter 4

Measuring running time

As we have discussed in Chapter 2, sampling-based running time prediction follows the classic machine learning mechanism, which usually consists of data pre-processing, model selection, model building and evaluation. In the data pre-processing stage, we aim to obtain quality data points that can be used not only by the base estimator to build its mathematical model, but also in the evaluation stage to test the prediction performance of the estimator. When measuring a program's running time, there are two important factors that affect the data quality: one is the point estimation method that we use to estimate the data value; the other is the tool used for the measurement. In this chapter, firstly we focus on the methods used to estimate the running time data point value, answering questions such as why we use an estimate based on data points from multiple runs instead of a single run, and why statistics of interest about the data points provide a more informative picture than a single observation. Secondly, we describe the programming tools we employed to measure the running time of an algorithm written in Java. At the end of this chapter, a brief discussion of the limitations and difficulties of these methods is given.

4.1 Is a single observation good enough?

The basic idea of measuring an algorithm's running time is trivial. Figure 4.1 shows the logic. We simply calculate the difference between the start time and the completion time, which is usually called the elapsed time. For example,

```
Procedure MeasureRunningTime
    startAt ← getCurrentTime()
        use an algorithm to do a task
    completeAt ← getCurrentTime()
    timeElapsed ← completeAt - startAt
End
```

Figure 4.1: Pseudo-code for measuring an algorithm's running time

algorithm A starts task k at 10:00:00AM and finishes its job at 10:01:00AM the same day. We say the algorithm's running time for this instance is one minute or 60 seconds. However, can we conclude that the running time of algorithm A for doing task k is always one minute? To answer this question, we can get algorithm A to do the same task again, and see whether the elapsed time is one minute. Table 4.1 lists five running time measurements of WEKA's J48 decision tree algorithm building its model on a data set. The experiment was carried out on an 3GHz Intel P4 PC running Ubuntu Linux 8.1 with only the software essential to run the experiment installed. The running time data of runs 1, 2, 3 and 5 are quite close to each other but run 4 is much longer. However, none of them are identical. We can see that the running time of an algorithm taking the same input instance varies from run to run. This fact is due to many reasons, such as the noise caused by memory management, caches, compiler optimization operations, and CPU usage by other programs. Based on empirical experiments like this one, we conclude that a single observation of the running time is not a proper estimate of the true running time value.

Next, we consider in statistical context, and under which conditions, a statistic of interest, such as the sample mean, provides a more reliable value than a single observation.

Run ID	Elapsed time in nanoseconds (1 nanosecond = 10^{-9} second)
1	901491
2	901212
3	898418
4	1334217
5	924119

Table 4.1: Five running time measurements for the J48 algorithm building its model on a data set.

4.2 Why use the sample mean?

A number of measurements are taken of some quantity, for example, a program's running time, in order to obtain an estimate of the quantity μ being measured. If the n measured values are $x_1, \ldots, x_n$, a common recommendation is to estimate μ by their mean

$$\bar{x} = \frac{(x_1 + \cdots + x_n)}{n}.$$

To answer the question asked in the section title, we first apply two data analytic methods. We use the least-squares approach and the sum of residuals approach to examine the mean. Suppose the true running time value being measured is the value t. The sum of squared differences $\sum(x_i - t)^2$ is the least-squares estimate of μ. That means the least-squares estimate is the value minimizing the sum of the squared residuals, the residuals being the differences between the observations x_i and the estimated value. Since we have the identity (Lehmann & Casella, 1998)

$$\sum(x_i - t)^2 = \sum(x_i - \bar{x})^2 + n(\bar{x} - t)^2,$$

we can see that on the right side t is not involved in the first term, and the second term can be minimized by $t = \bar{x}$.

Second, we use the sum of residuals method, in which the principle is to ask for the value t for which the sum of the residuals is zero, so that the positive

and negative residuals are in balance. The condition on t is

$$\sum(x_i - t) = 0.$$

Again, it is easy to see that $t = \bar{x}$.

The two approaches derive the mean as a reasonable descriptive measure of the center of the observations. However, they can not justify $\bar{x}$ as an estimate of the true value μ, since no explicit assumption has been made connecting the observations x_i with μ (Lehmann & Casella, 1998). To establish such a connection, we need to assume that the observations are the values taken on by random variables that follow a joint probability distribution, ρ, belonging to some known class P (Lehmann & Casella, 1998; Moore & McCabe, 1999; Christensen, 2001; Spiegel & Stephens, 2008). The distributions are indexed by a parameter, say μ, taking values in a set, Ω, so that

$$P = \{\rho_\mu, \mu \in \Omega\}.$$

Under this probability model, all we need to do is specify a reasonable value for μ. Here, μ can be viewed as a summary of the information provided by the data. That is, we estimate μ by a function $g(X)$, where X are the data points. The function $g(X)$ could be the mean function, or any statistics of interest obtained from the data. The assumptions above can be regarded as using the classical inference theory.

In the perspective of Bayesian analysis, μ is a random variable with a known distribution. In our running time prediction context, there is no problem if we see μ is itself a random variable since we know that the running time of an algorithm taking the same input instance varies from run to run. However, the second part of the assumption of the Bayesian approach is not as suitable in our context because we do not know the prior distribution of μ. For simplicity, and the reasons given below, we assume the prior distribution is normal.

One reason is that the normal distribution is well studied and commonly used, and has many convenient statistical properties and mathematical results. In the case of normal distribution, the xs (data points) have a variance of σ^2, and the variance of the mean $\bar{x}$ is σ^2/n, so the expected squared difference between the mean $\bar{x}$ and μ is only $1/n$ of what it is for a single observation (Lehmann & Casella, 1998). Another reason is that if the xs do not have a normal, but a Cauchy distribution (Papoulis, 1984; Spiegel, 1992)—also known as the t distribution with one degree of freedom—which has no mean or variance, then the distribution of $\bar{x}$ is the same as that of a single x_i (Lehmann & Casella, 1998). If that is the case for the running time data, then taking several measurements and averaging them as dictated by assuming a normal is the same as just taking a single observation.

For these reasons, we decided to assume the prior distribution for μ, is normal. Note also that, even if the population distribution, in our case the running time distribution, is not normal, as the sample size increases, the distribution of $\bar{x}$ gets closer to a normal distribution. And this holds true no matter what shape the population distribution has, as long as the population has a finite standard deviation (Moore & McCabe, 1999).

Let us now consider an example with real data. Figure 4.2 shows normal probability plots with distribution fit for J48's running time when the number of runs is 10, 100, 1000 and 10,000. For each run, J48 builds its model on a data set with 100 instances, six numeric attributes, three nominal attributes and one nominal class attribute. The confidence interval is set to be 95%. We can see that when the number of runs is 10, the data is very close to normal. This empirically suggests that in the running time prediction context, the sample mean may be a proper estimate of the true running time. However, when the number of runs is 100, the data shows a clear departure from the theoretical normal model. This very interesting result suggests that we can not say the running time data are truly from a normal distribution. Next we see that in the 1000 runs and the 10,000 runs cases, the patterns are similar: the data is

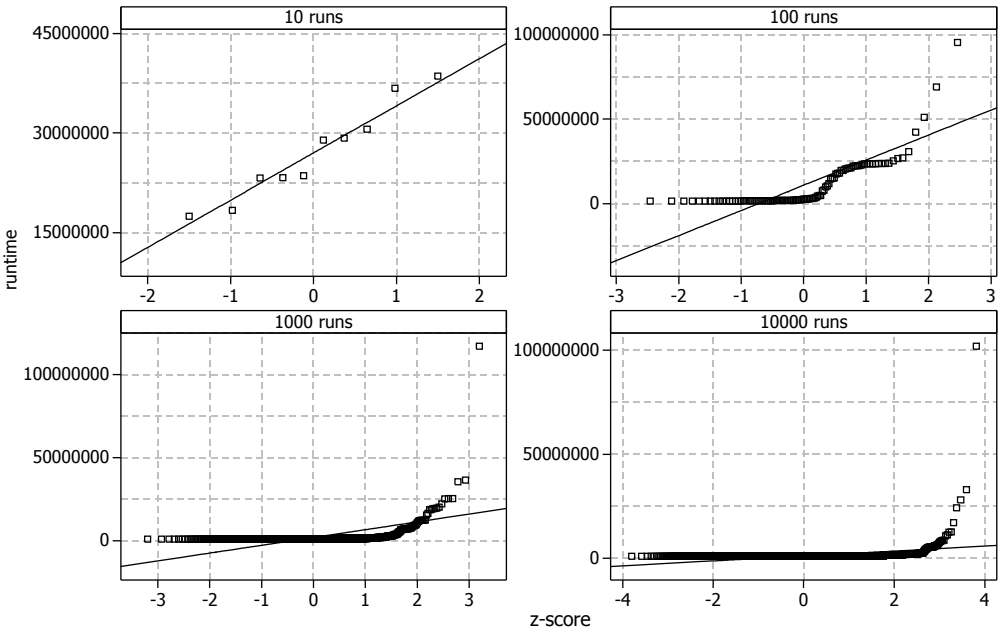


Figure 4.2: Normal probability plot with distribution fit for J48's running time when the number of runs is 10, 100, 1000 and 10,000. For each run, J48 builds its model on a training data set with 100 instances, 6 numeric attributes, 3 nominal attributes and 1 class attribute. The confidence interval (CI) = 95%

Number of runs	Mean	Anderson-Darling statistic
10	26965463	0.293
100	10799027	10.518
1000	1827616	292.638
10000	988342	3365.080

Table 4.2: Anderson-Darling statistics for the running time data in Figure 4.2

bent up at the right, showing right skewness. This is due to the outliers. If we remove the outliers, the data is close to normal.

Table 4.2 gives the Anderson-Darling (Anderson & Darling, 1952) test results associated with the data in Figure 4.2. The Anderson-Darling statistic measures how well the data follow a normal distribution. The better the distribution fits the data, the smaller this statistic will be. We can see that as the number of runs increases, the value of the Anderson-Darling statistic increases as well. This is an interesting phenomenon because it suggests that the observed running time data are not from a normal distribution: a point estimation method based on the normal distribution assumption may not be appropriate. This result is counter-intuitive.

Based on the above theoretical considerations, we conclude that, the mean

provides a more reliable value than a single observation when the running time data are from a normal distribution. However, the true shape of the running time data may have a more advanced distribution. Nevertheless, in this work, we use the mean as the point estimation method in all experiments described in Chapter 5.

4.3 Measuring the running time for an algorithm written in Java

As the goal is to build a running time estimator based on a few sample points (running time data in this case) to predict the running time of an unobserved program-execution time, the quality of the data—the measurement accuracy of the running time for training the model is crucial to the performance of a running time estimator. For that reason, we examine the prediction performance of each running time estimator not only on different algorithms and input data sets, but also on different measurement methods. Chapter 5 contains a detailed discussion of the experiment results. In this section, we focus on the running time measurement methods.

The Java programming language provides some built-in application programming interfaces (APIs) that can be used to get a value for the *getCurrentTime* function stated in Figure 4.1. In addition to these API methods, there are some other measurement methods which are based on the Java native interface (JNI) interacting with the time function of the underlying system. In the preliminary research stage, we explored five different measurement methods including the Java built-in APIs, JNI with C, and a third-party benchmark tool to measure the running time of 41 WEKA machine learning algorithms. Here is a very brief summary of these five methods:

- Method A—Use Java's *System.nanoTime()* to get running time in nanosecond, or *System.currentTimeMillis()* to get running time in milliseconds;

- Method B—Use Java's *getCurrentThreadCpuTime()* method of the *ManagementFactory* class;
- Method C—Use a benchmark tool (Boyer, 2008) to measure the elapsed time;
- Method D—Use a benchmark tool (Boyer, 2008) to measure the CPU time;
- Method E—Use JNI to call C's *clock()* function.

4.4 Measurement experiment

Table 4.3 gives an example running time data sheet for WEKA's J48 decision tree algorithm building its model on a data set of size 1000. The running time data were measured using the five methods from above. We can see that the running time data measured by methods A and B are close, the means are all about 0.03 second. The running time data measured by methods C and D are also close, with both means about 0.0009 second. We discuss the values measured by method E in the next section.

In this experiment, the mean running time calculated based on the data measured by methods A and B is about 33 ($\frac{0.03}{0.0009}$) times longer than that measured by using methods C and D. Does this mean the benchmark tool is a more accurate running time measurement instrument compared with simply measuring the elapsed time as implemented by methods A and B? To answer this question, we run another experiment on a larger data set and the observed running time data is given in Table 4.4. This time, we can see that the mean running time values calculated based on data measured by the five methods are all about four seconds. For this case, it is hard to say which measurement method is better.

Assume an algorithm completes its task in a very short time (as in the first case), where the elapsed time is τ, the true running time is supposed to be t,

Method A		Method B		Method C	
ID	t	ID	t	ID	t
1	86983355	1	70000000	1	1034095
2	29210490	2	30000000	2	855668
3	23310956	3	30000000	3	919490
4	20025397	4	10000000	4	937744
5	15512609	5	10000000	5	883249
mean	35008561	mean	30000000	mean	926049
	(0.035 s)		(0.030 s)		(0.00092 s)

Method D		Method E		
ID	t	ID	t	
1	1031250	1	1000000000	
2	881835	2	1000000000	
3	899414	3	1000000000	t is in nanoseconds
4	851562	4	1000000000	
5	1027343	5	1000000000	
mean	938280	mean	1000000000	
	(0.00093 s)		(1 s)	

Table 4.3: Results of using different running time measurement methods to measure the running time data of WEKA's J48 decision tree algorithm building its model on a data set with three nominal attributes, six numeric attributes, one class attribute and 1000 instances

the noise caused by compiler optimization or all other factors is e. We have $\tau = t + e$, and define T as a time length. It was observed that if $\tau < T$ then e is much greater than t ($e \gg t$), otherwise $e \ll t$. We do not know what exactly T should be, but our experiment suggests T is about 1000 ms. In the situation $\tau < T$, the running time of the algorithm itself only contributes to a small part of the total elapsed time. Therefore, we know that the running time data measured by certain methods, such as A and B, can be much longer than the actual running time when the measured elapsed time τ is less than T.

4.5 Time unit and resolution

The finest time unit provided by the Java programming language version 1.6 is one nanosecond. The running time data in Table 4.1 were measured by using method A. We can see that the running time data are quite different in the nanosecond resolution. In the last section, we discussed that this is due

Method A		Method B		Method C	
ID	t	ID	t	ID	t
1	4847524758	1	4840000000	1	4133831192
2	4297341591	2	4270000000	2	4100922994
3	4301882594	3	3870000000	3	4157013278
4	4177055448	4	3870000000	4	4032186658
5	4346481635	5	3870000000	5	4217231554
mean	4394057205	mean	4144000000	mean	4128237135
	(4.39 s)		(4.14 s)		(4.12 s)
Method D		Method E			
ID	t	ID	t		
1	3838000000	1	4000000000		
2	3882000000	2	5000000000		
3	3836000000	3	5000000000		
4	3921999999	4	4000000000		
5	4074000000	5	4000000000		
mean	3910400000	mean	4400000000		
	(3.91 s)		(4.40 s)		

t is in nanoseconds

Table 4.4: Results of using different running time measurement methods to measure the running time data of WEKA's J48 decision tree algorithm building its model on a data set with three nominal attributes, six numeric attributes, one class attribute and 30000 instances

to the tool used for monitoring the running time, which is not sophisticated enough to read the true running time. But if we had the perfect measurement tool, could we conclude that the running time data must all be the same? The answer is: "It depends". If we convert the values of running time data in Table 4.1 into the values shown in Table 4.5, we can see that they are not the same at the nanosecond resolution, thus we do not have the perfect tool for this resolution. However, for the second (time unit) resolution, we had the perfect tool, because all values are the same. This fact may motivate one to use the finest resolution possible. However, Boyer (2008) pointed out that the actual time resolution is not only dependent on the measurement instrument, but also depends on the underlying operating system and hardware. Table 4.6 lists the resolution levels provided by different operating systems. We can see that, although the Java programming language supports the nanosecond level of time resolution, most operating systems can support time resolution only at a ten-milliseconds level. Therefore, in this work, measurement methods A

	Elapsed time nanosecond = 10^{-9} second microsecond = 10^{-6} second millisecond = 10^{-3} second			
Run ID	in nanoseconds	in microseconds	in milliseconds	in seconds
1	901491	901	1	0
2	901212	901	1	0
3	898418	898	1	0
4	1334217	1334	1	0
5	924119	924	1	0

Table 4.5: Five running time measurements for the J48 algorithm building its model on a data set, running time data in four time units

Resolution	System
55 ms	Windows 95/98
10 ms	Windows NT, 2000, XP single processor
15.625 ms	Windows NT, 2000, XP dual processor
∼15 ms	Windows Vista
10 ms	Linux kernel 2.4
1 ms	Linux kernel 2.6
1 ms	Mac OS X

Table 4.6: Time resolution provided by different operating systems, adapted from (Boyer, 2008)

and B were employed when measuring the running time data for 41 WEKA machine learning algorithms. We did not use the benchmark tool (used by methods C and D) for the experiments described in Chapter 5, because we observed that the running time cost of the benchmark tool itself is too high to be used as a practical running time estimator. Method E, which uses the JNI to call C's *clock()* function, gives no better resolution than method B, so we did not use it, either.

4.6 Conclusions

As we have seen, measuring true running time by running an algorithm for a certain input size is a difficult problem in two regards. One aspect is, we need a measuring tool that is able to obtain a relatively accurate running time. The other is that, we need to use a point estimation method to estimate the true

running time of an algorithm. Both have a great influence on the prediction performance of a running time estimator. Therefore, in the evaluation stage, the running time estimators are compared under the same configurations, and for each configuration we make sure the training data sets, the tool used to measure the running time and the point estimation method, are same. In the running time data generation procedure applied for this work, for each measurement method (methods A and B), five runs of measurements were obtained for each algorithm-execution instance. The proposed running time estimators are compared separately for each measurement method, so that random errors caused by the measurement tools do not affect the evaluation results.

Chapter 5

Experimental results

In the previous chapters, we have discussed how to use interpolating curve fitting methods for extrapolation problems, and considered in detail the 11 running time estimators proposed in this thesis. This chapter presents experimental results on the prediction performance of those estimators, predicting the running times of 41 WEKA machine learning algorithms when building models on an artificial data set. In Section 5.7, we also consider the performance of the 11 estimators when used for predicting the running time of WEKA's SMO classifier on nine real world data sets.

5.1 Environment used for the experiments

The running time data were obtained on an Apple computer system with the following hardware and system specifications:
Hardware

- Processor: Intel Core Duo 1.66 GHz
- Memory: 1 GB

Software

- Operating system: Mac OS X 10.5.3 Leopard
- WEKA: version 3.7

The average CPU usage of other system processes while running an experiment was less than 3%.

```
@relation weka.datagenerators.classifiers.classification.Agrawal
@attribute salary numeric
@attribute commission numeric
@attribute age numeric
@attribute elevel {0,1,2,3,4}
@attribute car {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}
@attribute zipcode {0,1,2,3,4,5,6,7,8}
@attribute hvalue numeric
@attribute hyears numeric
@attribute loan numeric
@attribute group {0,1}
@data
110499.735409,0,54,3,15,4,135000,30,354724.18253,1
30372.275651,16722.784451,80,0,5,3,135000,10,481605.899589,0
119159.651677,0,49,2,1,3,135000,22,122025.085242,1
20000,52593.636537,56,0,9,1,135000,30,99629.621457,1
33167.375416,65126.594875,26,1,18,6,135000,3,475809.177725,0
...
```

Figure 5.1: Excerpt of the artificial data set used for getting the running time data of 41 WEKA machine learning algorithms, in WEKA's attribute-relation file format (ARFF) (Witten & Frank, 2005)

5.2 Running time data sampling

Given a training data set A of size N and a machine learning algorithm M, our goal is to predict the running time of M building a model on A based on data points obtained by using M on a small percentage of A. For instance, say $N = 100$, and we want to use only up to 5% of training data set A. One possible approach is to run algorithm M on five subsets of A, where the sizes of those five subsets could be 1, 2, ..., 5. This way we will have five running time data points.

One question is, how to sample these subsets of A? We have tried two sampling strategies: random sampling and additive sampling. The idea of random sampling is simple and works as follows: say we want five subsets of A with sizes of 1, 2, ..., 5. For subset 1, we randomly choose one instance from the training data set A, for subset 2 we randomly choose two instances, and accordingly for subset 3, subset 4 and subset 5, we randomly choose three, four, and five instances from A.

In contrast, the additive sampling strategy works as follows. Again, we want five subsets of sizes 1, 2, 3, 4 and 5. We first randomly choose one instance from A for subset 1. For subset 2, we randomly choose only one instance from A, and add the instance in subset 1 to subset 2 to get two instances. Accordingly, for subset 3, we have the instances in subset 2 plus one randomly selected from A. In the same way, subset 4 consists of subset 3 plus one and subset 5 of subset 4 plus one. When randomly selecting an instance and adding it to a set, the chosen instance could already be in the set. When this happens, we choose another instance, and make sure there are no identical instances in a set. We found that the additive sampling strategy makes the runtimes for different sizes more directly comparable and hence makes it easier to fit the curve. Therefore in this work, the additive sampling strategy was employed when sampling running time data for the 41 WEKA machine learning algorithms.

As mentioned in Chapter 4, for each input size, five measurements were acquired for calculating a statistic of interest (data point estimation). We first apply the $1.5 \times IQR$ (interquartile range) criterion for outlier detection to the data, and use the mean of the measurements with outliers removed as the data point value. Table 5.1 shows the running times observed from five runs of a WEKA machine learning algorithm training on a data set. In the first row, the largest observation ($measure_1$) is a suspected outlier. In IQR outlier detection, the interquartile range IQR is defined as the distance between the first and third quartiles: $IQR = Q_3 - Q_1$. Using the data in Table 5.1 as an example, $IQR = 22 - 20 = 2$, then $1.5 \times IQR = 3$. Any values below $20 - 3 = 17$ or above $22 + 3 = 25$ are seen as outliers. It can be seen that $measure_1$ (4029 ms) and $measure_4$ (69 ms) are two outliers based on the $1.5 \times IQR$ criterion, because both are greater than $Q_3 + 1.5 \times IQR$. Therefore they are removed from the observations before computing the mean.

In running time observation experiments, we found that the Java Virtual Machine (JVM) needs some time (usually under 1000 ms) to "warm up" before

working on an actual task. The "warm up" procedure is due to the real time optimization applied by the JVM to find an optimized code interpretation strategy for the underlying program code. The running time cost of this "warm up" procedure usually contributes to the time cost of the first running time observation. Therefore, an outlier detection method such as the IQR detection scheme needs to be applied.

The running time data used for the experiments were obtained by running the 41 WEKA machine learning algorithms on an artificial data set consisting of six numeric attributes, three nominal attributes and one class attribute. The data was generated using one of WEKA's data set generator tools. Detailed parameter settings for the generator and the data set can be found in Appendix C. Figure 5.1 shows the structure of the data set.

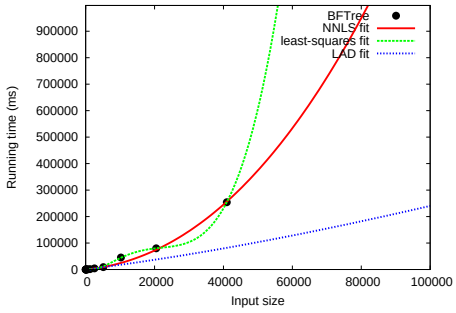
5.3 Running time data sets

Two different running time measurement methods (Section 4.3) were used when acquiring the running time data for the 41 WEKA machine learning algorithms. Consequently, we have two collections of running time data sets. One was obtained using method A (Section 4.3), another using method B (Section 4.3). We found that the prediction performance of the 11 estimators is similar for both running time data collection methods. The experimental results shown in this chapter are based on the first method. Experiments and results based on the second measurement method can be found in Appendix D.

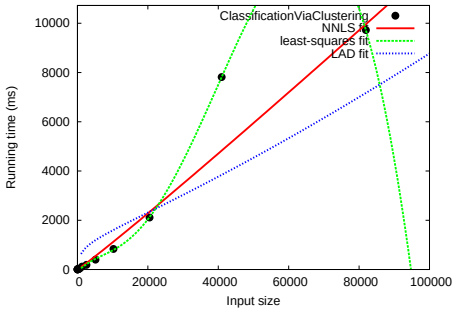
Table 5.1 shows a sample running time data file. The running time values were measured using method A (Section 4.3). All 41 WEKA machine learning algorithms examined in this work were used to generate data files like this one.

Input size	$measure_1$	$measure_2$	$measure_3$	$measure_4$	$measure_5$
10	4029	20	21	69	22
20	27	21	20	31	111
40	30	21	24	35	23
80	33	31	39	51	30
160	85	59	111	68	45
320	129	247	110	132	94
640	302	401	695	352	280
1280	988	954	959	1093	1017
2560	3666	3607	3831	3773	3810
5120	17219	16619	18081	17054	17093
10240	74496	74786	72584	72850	72007
20480	303874	308036	303687	300130	310681
40960	1303578	1309416	1311123	1308362	1317133
81920	5544938	5500496	5340639	5391164	5524745

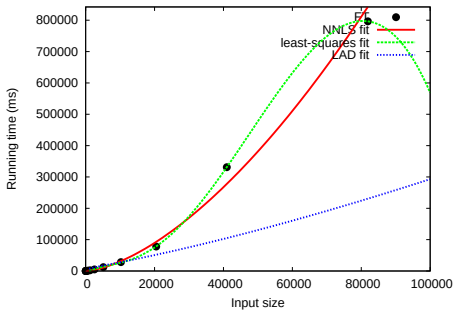
Table 5.1: A sample running time data file obtained using the SMO algorithm (WEKA implementation of support vector machine learning). For each input size, five running time measurements were obtained. Values are in milliseconds. The algorithm was run on a data set consisting of six numeric attributes, three nominal attributes and one class attribute



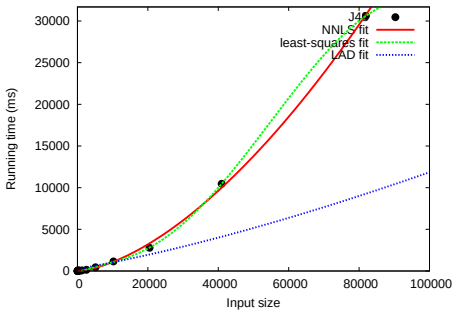
(a) BFTree



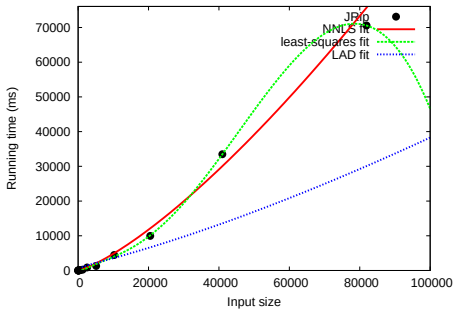
(b) ClassificationViaClustering



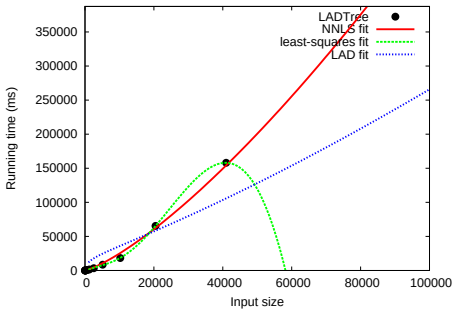
(c) FT



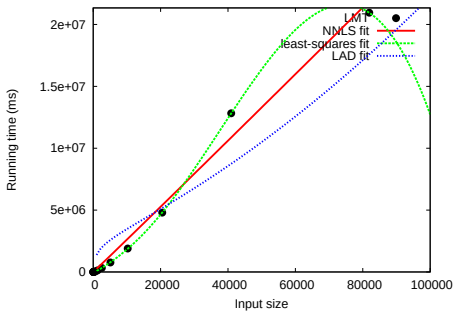
(d) J48



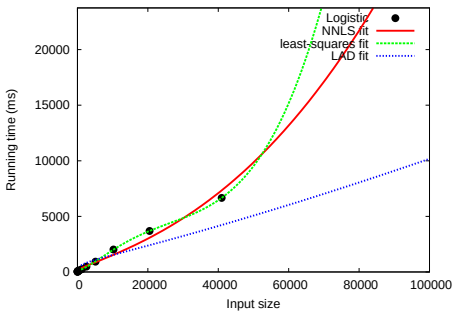
(e) JRip



(f) LADTree

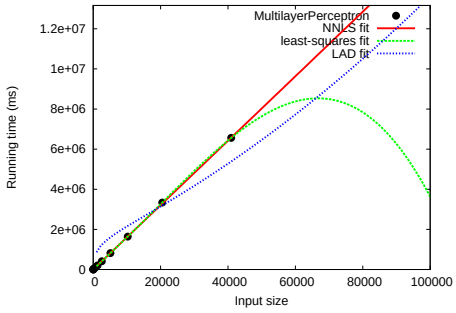


(g) LMT

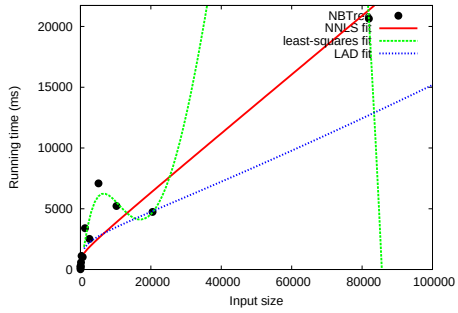


(h) Logistic

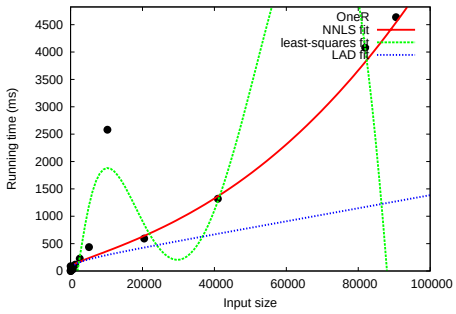
Figure 5.2: Running time data curve fitting



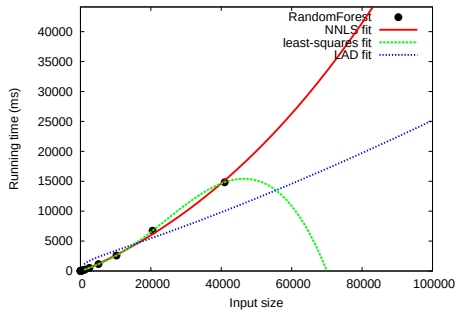
(a) MultilayerPerceptron



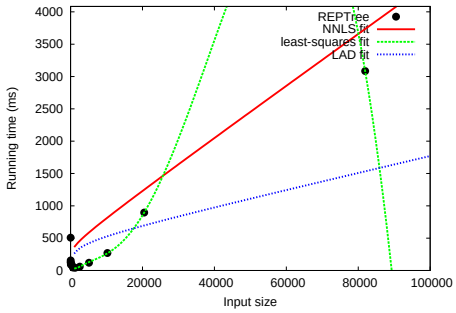
(b) NBTree



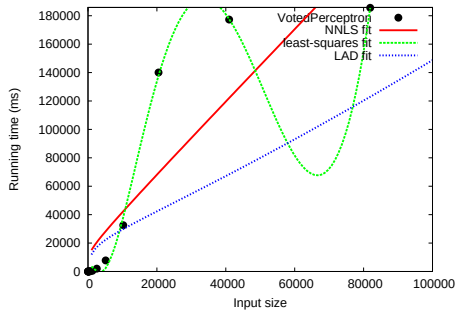
(c) OneR



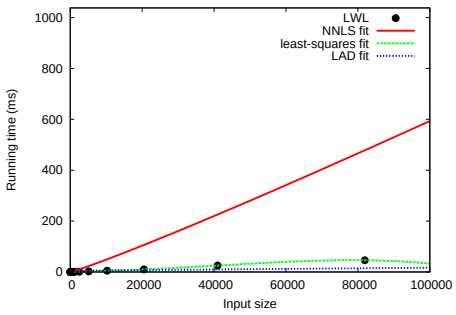
(d) RandomForest



(e) REPTree



(f) VotedPerceptron



(g) LWL

Figure 5.3: Running time data curve fitting

5.4 Curve fitting using least-squares, LAD and NNLS

In Chapter 2, we have discussed three curve fitting algorithms: least-squares, LAD and NNLS. Before we start describing the evaluation results for the estimators using these curve fitting algorithms, it is illustrative to consider how these algorithms fit running time data obtained by monitoring the running times of 41 WEKA machine learning algorithms on an artificial data set (six numeric attributes, three nominal attributes and one class attribute with two classes). In this experiment, the following linear regression model was used

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \beta_6 X_6,$$

$$X_1 = n,$$

$$X_2 = log(n),$$

$$X_3 = nlog(n),$$

$$X_4 = n^{1.5},$$

$$X_5 = n^{2.0},$$

$$X_6 = n^{2.5},$$

where n is the input size.

Figures 5.2 to 5.3 show some fitted curves obtained using the least-squares, LAD and NNLS algorithms with the above linear model, which are selected from the figures in Section E.1 of Appendix E. From the figures in Section E.1 of Appendix E, we can see that in most cases, the least-squares and the NNLS curves are close, and fit the data well. In some cases, the least-squares algorithm fits the data very well, but its curves show clear nonlinear shapes, such as Figures 5.2 (a), (b), (c), (d), (e), (f), (g), and (h), Figures 5.3 (a),

(b), (c), (d), (e) and (f). This implies that the least-squares fit is good at interpolation, but not extrapolation. The NNLS algorithm holds promises as the fitted curve is monotonic. However, in some cases, such as Figures 5.3 (e), (f) and (g), it does not fit the data well. The LAD curves do not fit each data point closely; however, they always satisfy the monotonic assumption.

Section E.2 of Appendix E shows fitted curves for those three algorithms on running time data obtained using the measurement method B described in Section 4.3.

In the following sections, we will consider the experimental results on the prediction performance of the running time estimators based on these curve fitting methods.

5.5 Evaluation by examining the absolute error of each prediction

In total, we have the running time data for 41 WEKA classifiers and 11 running time estimators. To see the prediction performance of each individual estimator compared with each other, the first evaluation method we employ is the absolute error (absolute difference) $\triangle x \equiv |x_0 - x|$, where x_0 is the predicted value, and x the actual value. In the running time prediction context, we assume we do not know the actual value of the true running time. Therefore, we use $\bar{x}$, the mean of the observed values as an alternative.

Assume we have an input data set A of size n, a machine learning algorithm C, and three estimators, $E1$, $E2$, and $E3$, giving p_1, p_2 and p_3 respectively as the predictions of the running time of C building its model on A. If $E1$'s prediction p_1 results in the lowest absolute error among these three estimators, we say estimator $E1$ won this experiment instance.

Algorithm 5 shows the pseudo-code of the evaluation algorithm used to calculate and examine the absolute error of each prediction.

Let us consider a case study designed to compare prediction performance

Algorithm 5 Pseudo-code for calculating prediction performance based on absolute error

```
For Each test observation instance

  tm = testObservation.meanOfRunningtime

  For Each estimator

    estimatedRunningtime = estimator.predict(testObservation)

    error[estimator] = ABS(estimatedRunningtime - tm)

    IF estimatedRunningtime <= 0
      error[estimator] = Double.Max
    End IF

  End For

  For Each estimator

    IF error[estimator] == MIN(error)
      numOfWins[estimator] += 1
    End IF

  End For

End For
```

between estimators, based on WEKA's SMO algorithm. The setup considered is as follows:

- Machine learning algorithm M: SMO
- Training set S for M: The artificial data set consisting of six numeric attributes, three nominal attributes and one class attribute.
- Estimators for testing: PSLR, BC, LADDER, LsF, LsR, LsSeq, LadF, LadR, nnlsF, nnlsR, OneTest
- Evaluation method: Number of wins based on the absolute error

To fairly compare between estimators, the training data points used by each estimator to construct its own estimation model need to be the same. Then, given testing data points, the predictions of each estimator are compared based on the evaluation method. In this case study, the absolute error criterion is employed. We use the first 11 (up to input size = 10240) running time data points in Table 5.1 as the training data points, and the remaining three data points as the testing data points.

Table 5.2 shows the running time predictions of the 11 estimators predicting on the three testing data points. Values ending with a "•" indicate a win by the smallest absolute error. We can see that the estimator LADDER (based on ladder transformations) wins for all three testing instances. However, although estimator LADDER outperforms others in this particular case study, this does not mean LADDER will also win when testing on the running time data of a set of machine learning algorithms.

Tables 5.3 and 5.4 show the results of examining the 11 estimators for all 41 WEKA machine learning algorithms using the same experimental setup as in the above case study. We can see that estimator LADDER has the highest number of wins (18), comprising about 20% of all 123 tests (three testing data points for each of the 41 algorithms). However, although LADDER wins for this particular setup, that does not mean LADDER will necessarily win when

Input size	Observed (mean)	PSLR	BC	LADDER	LsF	LsR	LsSeq	LadF	LadR	nnlsF	nnlsR	OneTest
20480	305281.6	42160	139568	314237•	200552	253473	293835	224394	158673	6E12	153888	224394
40960	1308119.7	93847	293474	1347136•	890177	784341	1103598	1027030	341781	3E13	352257	1027030
81920	5460396.4	208898	611026	5775267•	3658290	2250898	3949427	4816401	734001	2E14	799476	4816401

Table 5.2: Predictions of the 11 estimators for three testing data points. Values are in milliseconds

Algorithm	PSLR	BC	LADDER	LsF	LsR	LsSeq	LadF	LadR	nnlsF	nnlsR	OneTest
RandomForest	0	0	3	0	0	0	0	0	0	0	0
SMO	0	0	3	0	0	0	0	0	0	0	3
BFTree	0	0	0	0	0	2	1	0	0	0	1
DecisionStump	0	0	2	0	0	0	0	0	0	1	2
MultilayerPerceptron	0	0	0	0	0	0	0	0	1	0	2
ClassificationViaRegression	0	0	0	0	1	0	2	0	0	0	0
ConjunctiveRule	0	0	0	0	0	0	0	0	0	3	0
Logistic	0	0	0	0	1	0	0	2	0	0	0
J48	0	0	1	1	0	0	1	0	0	0	0
LADTree	0	0	1	0	0	0	2	0	0	0	0
DecisionTable	0	3	0	0	0	0	0	0	0	0	0
J48graft	0	0	0	0	3	0	0	0	0	0	0
StackingC	0	0	0	0	0	0	0	2	0	1	0
AttributeSelectedClassifier	0	0	0	1	1	0	0	1	0	0	1
ClassificationViaClustering	0	0	0	1	1	0	1	0	0	0	0
OneR	0	1	1	0	0	0	0	0	1	0	0
NNge	0	0	0	0	3	0	0	0	0	0	1
RBFNetwork	0	0	1	0	0	1	0	1	0	0	0
IB1	0	0	0	0	0	0	0	0	0	0	0
PART	0	0	2	0	0	0	1	0	0	0	0
ADTree	0	0	0	2	0	0	1	0	0	0	0
Stacking	0	0	0	0	0	0	0	3	0	0	0

Table 5.3: Predictive performance evaluation by counting the number of wins (based on absolute error) of the 11 estimators over 41 WEKA machine learning algorithms, Part 1

	PSLR	BC	LADDER	LsF	LsR	LsSeq	LadF	LadR	nnlsF	nnlsR	OneTest
RandomTree	0	0	0	0	0	0	0	0	0	0	0
LWL	0	0	0	0	0	0	0	0	0	0	0
AdaBoostM1	0	0	0	1	2	0	0	0	0	0	2
IBk	0	0	0	0	0	0	0	0	0	0	0
Bagging	0	0	2	0	1	0	0	0	0	0	0
Vote	0	0	0	0	0	0	0	0	0	0	0
BayesNet	0	1	0	0	0	0	0	1	0	1	0
FT	0	0	0	0	0	0	3	0	0	0	0
DTNB	0	2	0	0	0	0	0	1	0	0	2
Ridor	0	0	2	0	1	0	0	0	0	0	0
KStar	0	0	0	0	0	0	0	0	0	0	0
NBTree	0	1	0	0	0	0	1	0	0	1	0
LMT	0	0	0	0	2	0	0	0	0	1	0
NaiveBayes	0	0	0	0	0	0	0	0	0	0	0
ZeroR	0	0	0	0	0	0	0	0	0	0	0
REPTree	0	0	0	0	0	0	2	1	0	0	0
SimpleCart	0	0	0	0	2	0	0	0	0	1	0
VotedPerceptron	0	1	1	0	0	0	0	0	0	1	0
JRip	0	0	1	2	0	0	0	0	0	0	0
Total:	0 (0%)	8 (9%)	18 (20%)	7 (8%)	16 (18%)	3 (3%)	14 (15%)	11 (12%)	2 (2%)	9 (10%)	13 (14%)

Table 5.4: Predictive performance evaluation by counting the number of wins (based on absolute error) of the 11 estimators over 41 WEKA machine learning algorithms, Part 2

Setup ID	Training data points	Testing data points
1	{10, 20, 40, 80, 160} Size: 5	{320, 640, 1280, 2560, 5120, 10240, 20480, 40960, 81920}
2	{10, 20, 40, 80, 160, 320} Size: 6	{640, 1280, 2560, 5120, 10240, 20480, 40960, 81920}
3	{10, 20, 40, 80, 160, 320, 640} Size: 7	{1280, 2560, 5120, 10240, 20480, 40960, 81920}
4	{10, 20, 40, 80, 160, 320, 640, 1280} Size: 8	{2560, 5120, 10240, 20480, 40960, 81920}
5	{10, 20, 40, 80, 160, 320, 640, 1280, 2560} Size: 9	{5120, 10240, 20480, 40960, 81920}
6	{10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120} Size: 10	{10240, 20480, 40960, 81920}
7	{10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240} Size: 11	{20480, 40960, 81920}
8	{10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240, 20480} Size: 12	{40960, 81920}
9	{10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240, 20480, 40960} Size: 13	{81920}

Table 5.5: Different training/testing setups for evaluating the prediction performance of the 11 estimators over 41 WEKA machine learning algorithms, based on evaluating the absolute error

Rank	Estimator	Percentage of wins
1	LadR	24%
2	OneTest	16%
3	LadF	10%
4	LADDER	9%
5	nnlsR	8%
5	LsF	8%
5	LsR	8%
5	BC	8%
9	LsSeq	5%
10	PSLR	3%
11	nnlsF	2%

Table 5.6: A ranked list of the 11 estimators. Ranking positions are based on the prediction performance of the 11 estimators over 41 WEKA machine learning algorithms under 9 different training/testing setups, using absolute error as the evaluation criterion

the experimental setup is changed. For example, the number of training data points or the number of testing data points can be changed. Therefore, we will next show the prediction performance of the 11 estimators under several training and testing setups.

Table 5.5 shows the details of the training and testing data setups for this experiment. The numbers in the columns "Training data points" and "Testing data points" indicate the input size values of a simple running time experiment. The experiment counts the percentage of wins over all tests (based on the number of wins per setup) each estimator received for each of the nine setups. Table D.1 in Appendix D shows the results.

Figure 5.4 shows the prediction performance curves of the 11 estimators as the size of the training data increases from 5 to 13. The values for "percentage of wins" are calculated by counting the number of wins based on the absolute error evaluation, and then dividing by the number of tests performed for a particular size of training data. We can see that estimator LadR outperforms the other estimators when the size of the training data grows from five to

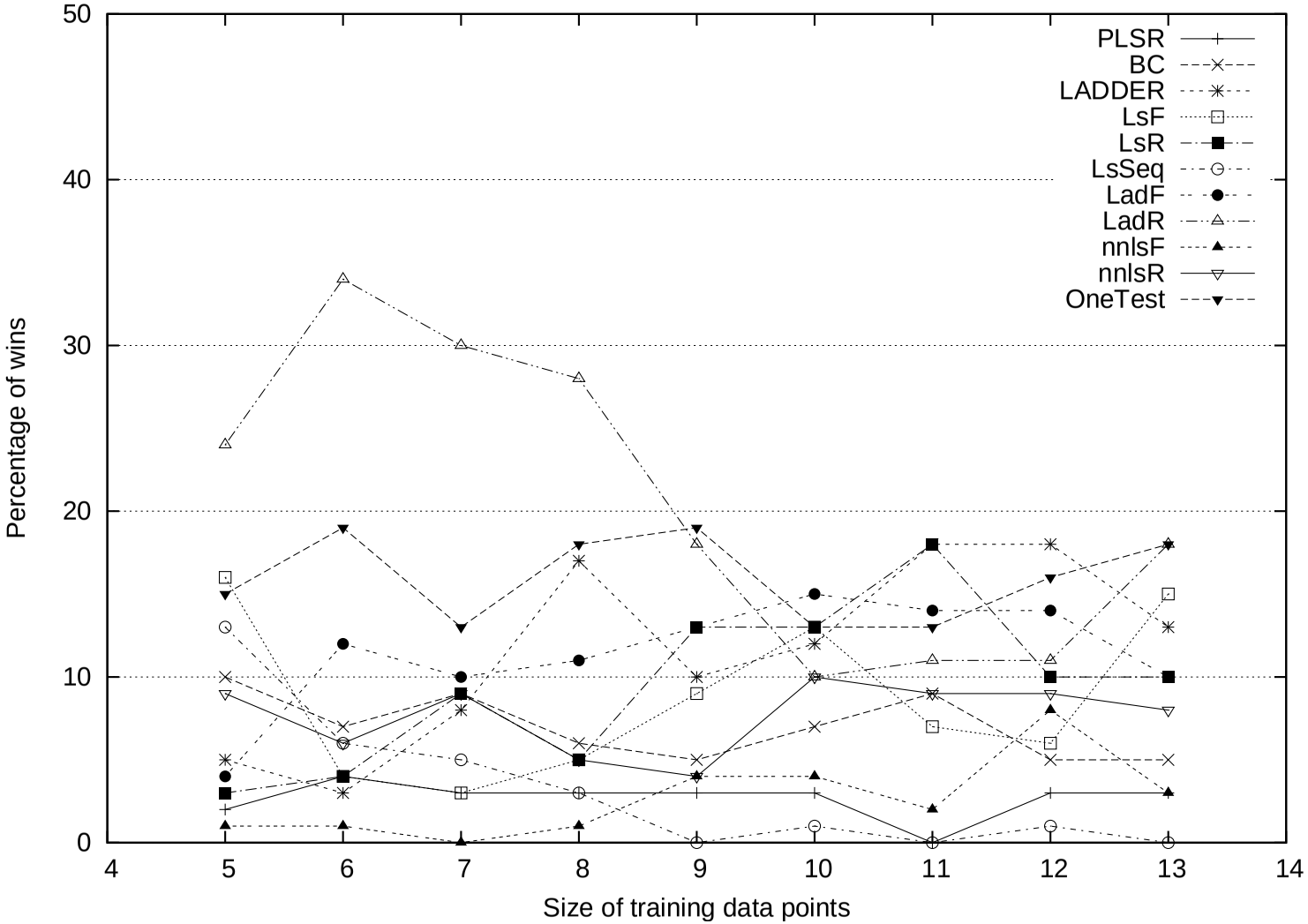


Figure 5.4: Prediction performance curves of the 11 estimators as the number of observations increases, evaluated by examining the absolute error, on 41 WEKA classifiers

eight, which means LadR's performance is good when the size of the training data points is small. Excluding estimator LadR, as the size of the training data points increases, we can see that the estimators OneTest and LadF outperform the other estimators. Another interesting point is that, when there are enough training data points—for example, as the size of the training data grows from 10 to 12—the estimator LADDER has the highest percentage of wins.

Table 5.6 shows a ranked list of the 11 estimators for this experiment, sorted by the percentage of wins. It can be seen that the estimators LadR (least absolute deviations based on a restricted linear regression model) and OneTest (regression meta learner) outperform the other estimators.

5.6 Evaluation by examining the quality of each prediction

In Table 5.6, we have a list of running time estimators ranked by percentage of wins based on the absolute error criterion, but we do not consider whether predictions are really qualitatively different. In this section, we consider a method that examines the estimated prediction quality of each of the 11 estimators using a discretized range of error values.

Algorithm 6 shows the pseudo-code of the evaluation algorithm that computes an estimated quality value for each prediction. The idea is that rather than focusing on the distance between a prediction and the observed value, we give a fixed distance value that indicates how close the prediction is to an observed value. The distance between the prediction and the observed value is divided into levels. The closer a level's boundary to the observed value, the smaller the distance value that the level will get.

In this way, if two estimators have predictions belonging to the same level, we say the quality of the predictions made by these two estimators is the same. Based on this, we repeat the experiment described in the last section, except the evaluation method is replaced by examining the estimated quality of

Algorithm 6 Pseudo-code for calculating prediction performance based on the estimated quality of each prediction

```
For Each test observation instance

  tm = testObservation.meanOfRunningtime

  For Each estimator

    error[estimator] = Double.Max

    estimatedRunningtime = estimator.predict(testObservation)

    IF estimatedRunningtime <= 1.5 × tm AND
         estimatedRunningtime >= 0.5 × tm
      error[estimator] = 150
    End IF

    IF estimatedRunningtime is a value between the upper and lower limits
         of the estimated population mean of testObservation in 95% CI
      error[estimator] = 95
    End IF

    IF estimatedRunningtime is a value between the upper and lower limits
         of the estimated population mean of testObservation in 90% CI
      error[estimator] = 90
    End IF

    IF estimatedRunningtime is a value between the upper and lower limits
         of the estimated population mean of testObservation in 70% CI
      error[estimator] = 70
    End IF

    IF estimatedRunningtime is a value between the upper and lower limits
         of the estimated population mean of testObservation in 50% CI
      error[estimator] = 50
    End IF

    IF estimatedRunningtime <= 0
      error[estimator] = Double.Max
    End IF

  End For

  For Each estimator

    IF error[estimator] == MIN(error)
      numOfWins[estimator] += 1
    End IF

  End For

End For
```

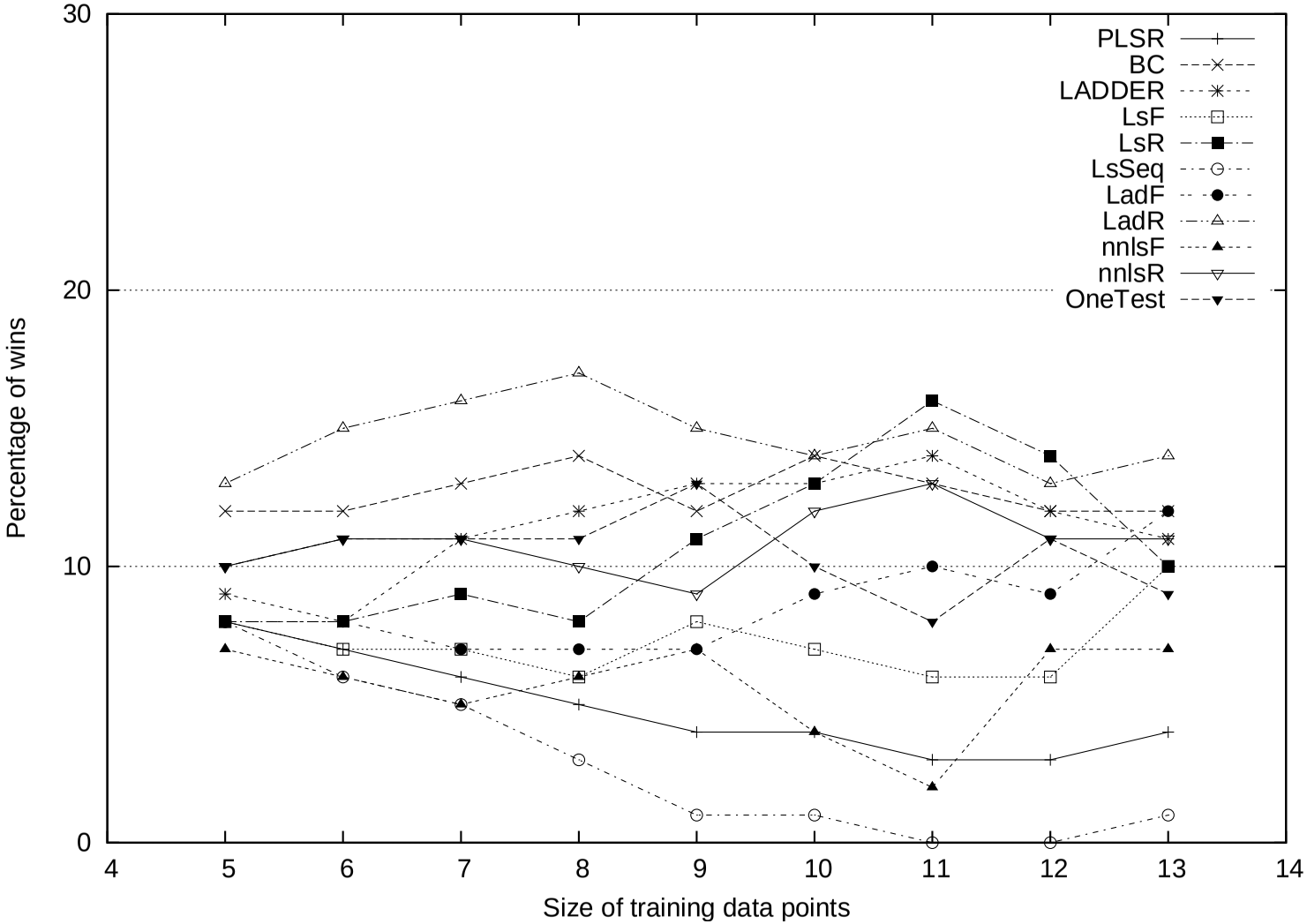


Figure 5.5: Prediction performance curves of the 11 estimators as the number of observations increases, evaluated by examining the quality of predictions of these estimators, on 41 WEKA classifiers

Rank	Estimator	Percentage of wins
1	LadR	15%
2	BC	13%
3	nnlsR	11%
4	OneTest	10%
4	LADDER	10%
6	LsR	9%
7	LadF	8%
8	LsF	7%
9	PSLR	6%
9	nnlsF	6%
11	LsSeq	2%

Table 5.7: A ranked list of the 11 estimators. Ranking positions are based on the experimental results of the prediction performance of the 11 estimators over 41 WEKA machine learning algorithms under nine different training/testing setups, using absolute errors as the evaluation criterion

each prediction. Figure 5.5 shows the prediction performance curves of the 11 estimators while the size of training data increases. The values for "percentage of wins" are calculated by counting the number of wins based on the estimated quality of each prediction divided by the number of tests performed for a particular size of training data. We can see that the estimators LadR and BC outperform the others when the training data sizes are small (from 5 to 8). After that, as the size of training data increases (from 9 to 13), the prediction performance of LsR, LADDER, nnlsR, LadF and OneTest is quite close. Overall, estimator LadR outperforms the other estimators.

Table 5.7 shows the ranked list for this experiment. It can be seen that the estimators LadR (least absolute deviations based on a restricted linear regression model) and BC (Box-Cox transformation) outperform the other estimators. Also, we can see that, on the whole, the prediction quality of the estimators nnlsR, OneTest, LADDER, LsR and LadF is actually very close.

Data set	# of instances	# of attributes	# of class labels
hypothyroid	3772	30	4
segment	2310	20	7
kr-vs-kp	3196	37	2
sick	3772	30	2
letter	20000	17	26
waveform-5000	5000	41	3
mushroom	8124	23	2
credit-g	1000	21	2
splice	3190	62	3

Table 5.8: Nine UCI data sets with detailed information

5.7 Evaluation using UCI data sets

In this section, we show experimental results obtained using the 11 estimators to predict the running time of the SMO algorithm building models on real world data sets. Nine UCI (Asuncion & Newman, 2007) data sets in total have been used for this experiment. Table 5.8 shows detailed information about those data sets. For each data set, each of the 11 estimators is applied using up to 16% (1%, 2%, 4%, 8% and 16%) of the full data set to generate five training data points.

The running time estimates for the SMO algorithm building models on these nine UCI data sets are given in Table 5.9. The values under the column "Mean RT" are the mean running time of SMO building model on a full data set, obtained by averaging running time values of five runs. The values under each estimator are their running time estimates for the SMO algorithm building model on the corresponding data set. These running time estimates of the 11 estimators are based on up to 16% of a full data set.

We counted the number of wins evaluated using the absolute error (see **Algorithm 5**) and the prediction quality (see **Algorithm 6**) criteria. For the first method, a win is indicated by a "•"; for the second method, a win is indicated by a "◦". From Table 5.9, it can be seen that the estimator LadF

Data set	Mean RT	PSLR	BC	LADDER	LsF	LsR	LsSeq	LadF	LadR	nnlsF	nnlsR	OneTest
hypothyroid	9089.4	1773.2	4440.7	5337.7○	188.7	20793.9	3846.4	11960.1●○	4884.2○	33115.7	4939.2○	11960.1●○
segment	1230.6	908.4○	1198.7●○	1194.2○	860.7○	770.5○	1194.2○	1947.6	1296.8○	4.5E11	368.7	1198.7●○
kr-vs-kp	5415.75	1433.0	2375.7	1104.3	709.1	-22549.3	2376.4	7667.6●○	2895.8○	2371.9	2348.8	1433.0
sick	2144.5	119.5○	399.5○	650.5○	49.3○	-2417.0○	349.5○	1059.5●○	425.5○	9.6E10○	486.6○	399.5○
letter	53870.0	27067.9○	43239.1○	22897.4	23912.2	46887.3●○	41108.7○	80030.6○	45086.3○	3.4E15	22350.5	45086.3○
waveform-5000	2796.6	499.0	1582.0○	1897.7●○	147.7	663.9	1390.6	4353.8	1691.8○	5.7E11	1025.2	1691.8○
mushroom	10927.0	4358.1	5440.2	5440.7	-33597.8	-11428.3	5440.7	16724.4	6589.8●○	5394.4	5424.8	4358.1
credit-g	1998.0	123.6	357.6	340.1	629.6	-1508.5	340.1	1053.5●○	429.9	2.4E9	255.7	255.7
splice	199722.0	128804.3○	99748.2	145.3	-61938.8	150353.0●○	85158.5	297433.8○	111442.2○	1.3E11	135611.1○	-61938.8
Total ●		0	1	1	0	2	0	4	1	0	0	2
Total ○		4	4	4	2	4	3	6	8	1	3	5

Table 5.9: The estimates of the 11 estimators predicting the running times of SMO building models on nine UCI data sets. A "●" indicates a win by using the absolute difference evaluation; a "○" indicates a win evaluated by the quality of each prediction. Values are in milliseconds

has the highest number of wins (four wins) in the absolute error evaluation test. In the prediction quality evaluation test, the estimator LadR has the highest number of wins (eight wins), and estimator LadF is second best with six wins; estimator OneTest is third with five wins. The running time prediction problems simulated by this experiment are similar to a real application of these estimators. We can see that in both evaluation methods, the performance of LAD-based estimators is superior to least-squares-based estimators. The performance of transformation based estimators, PSLR, BC and LADDER, is very close.

5.8 Conclusions

In this chapter, we first discussed experimental results based on the prediction performance of 11 estimators predicting the running times of 41 WEKA machine learning algorithms when used for building models on an artificial data set. Our results show that LadR is the best running time estimator among the 11 estimators proposed in this work, and it outperforms the other estimators in terms of two different evaluation strategies. The performance of the estimators LadF, OneTest, BC and LADDER is reasonably good for some specific algorithms, but not in general. In Section 5.7, we also considered experimental results obtained using the 11 estimators for predicting the running time of WEKA's SMO classifier on nine real world data sets. The results show that the estimators LadF and LadR outperform the other estimators based on the two evaluation strategies used.

Based on the experimental results, we conclude that in the running time prediction problems examined in this work, LAD (least absolute deviations) based running time estimators outperform least-squares-based estimators when both use the same underlying linear regression models.

Chapter 6

Conclusions

This thesis proposed eleven sampling-based running time estimators, and empirically evaluated their predictive performance. This was done by predicting the running times of 41 WEKA machine learning algorithms, building models on both artificial and real world data sets. Chapters 2 and 3 described the ideas underlying the construction of a sampling-based running time estimator. We explained that sampling-based running time prediction, by its very nature, is a function approximation problem. From a theoretical perspective, mathematical asymptotic analysis forms the foundation for sampling-based running time prediction methods. From a practical point of view, employing an appropriate running time measurement method, and applying statistical procedures to the observed data points is necessary. Chapter 4 focused on running time measurement tools and point estimation methods, and explained that running time measurement by itself is a very difficult problem in two regards. One is that we need a measuring tool that is able to obtain a relatively accurate running time; another is that we need to use a point estimation method to estimate the true running time of an algorithm. Both have a great influence on the predictive performance of a running time estimator. Experimental results for the estimators proposed in this thesis were presented in Chapter 5.

6.1 Main results and contributions of this thesis

Our experimental results show that the least absolute deviations (LAD) based running time estimators outperform the least-squares-based estimators for the

running time prediction problems discussed in this work. This finding appears to be novel, we did not find any research literature that has revealed a similar result. Another main result is that OneTest, the regression meta learner introduced in Chapter 3, is a competitive running time estimator, or regression meta learner, for running time prediction. It can be applied with arbitrary base estimators, so the idea can be applied whenever a new estimator is developed. We also found that, in general, when using the least-squares, the LAD, and the non-negative least-squares (NNLS) method as running time estimators, the predictive performance of these algorithms working on a restricted linear model (see Equation 3.3, Sections 3.7, 3.10, and 3.12) is better than that obtained from a "full" model (see Sections 3.6, 3.9, and 3.11).

Our experimental results show that the predictive performance of least-squares-based estimators can be strongly affected by the presence of noise in the training data points. It is clear that the least-squares fit does not always satisfy the monotonicity assumption (see figures in Sections E.1 and E.2 in Appendix E). Although the NNLS algorithm can force the least-squares algorithm to satisfy the monotonicity assumption by adding constraints to the linear regression model, the resulting fit is likely to be an upper bound, and in some cases far away from the observed data points (Section 5.4 and Appendix E).

This thesis makes several methodological contributions to research. These contributions are:

- the use of Box-Cox, and ladder transformations, as well as LAD-, and NNLS-based estimators for sampling-based running time prediction (Chapter 2);
- construction of a base-learner-independent regression meta learner for sampling-based running time prediction (Chapter 3);
- a method for modeling the running time function using augmented multiple linear regression models (both restricted and full versions) in terms

of the input sizes (Chapter 3);

- a wrapper-based feature subset selection method for a least-squares-based running time estimator (Chapter 3);
- a method for evaluating the predictive performance of running time estimators by examining the absolute error of predictions (Chapter 5);
- a method for evaluating the predictive performance of running time estimators by examining the quality of each prediction (Chapter 5);

The empirical contributions are:

- an experiment based on real world data sets used to evaluate the predictive performance of running time estimators (Chapter 5);
- experimental results for predicting the running time for 41 WEKA algorithms when used for building models on an artificial data set in nine different training data sampling setups (Chapter 5).

6.2 Future work

There are some questions for future research resulting from the issues addressed in this thesis. One question is whether the running time prediction methods proposed can be used to improve the accuracy of estimating the function $CPU(n)$ in Equation 1.1 of Section 1.2. If that is the case, then the accuracy of the cost model stated by Equation 1.1 can be improved.

An issue raised when using least-squares-based regression methods for the problem of sampling-based running time prediction is that least-squares-based regression is very good at fitting the observed data points, but not at extrapolation. One reason is that the returned polynomial running time function may not be a monotonic function, thus it may not satisfy the trend assumption. However, although we employed the NNLS algorithm to return non-negative predictions, the resulting running time function is still not competitive

with LAD-based algorithms. One avenue for future research is to investigate whether a least-squares-based algorithm can be extended to be more adequate for the monotonicity assumption inherent in the running time prediction problem.

To be of practical use, ideally, a running time estimator should finish its computation (including sampling and model construction) in a few seconds on a moderately powerful computer. In order to achieve this goal, the number of samples needs to be as small as possible. One direction for future research is to investigate how to compute an optimal sampling strategy for a machine learning algorithm and its input instance.

Feature selection is another avenue for future research. We did not apply the wrapper-based feature subset selection method to estimators other than least-squares based ones, so one direction for future research is to apply wrapper-based feature subset selection methods to LAD- or NNLS-based estimators.

As mentioned in Chapter 4, the accuracy of running time measurements of an estimator can be crucial when the sampling instance completes its computation in a very short time, such as under one second. One direction for future research is to design and implement a more sophisticated, ideally noise free and system-independent running time measurement tool.

This thesis proposed eleven sampling-based running time estimators. There are many other applications of these running time prediction methods, some of which have been discussed in Chapter 1. The experimental results presented in this thesis show that, with some care in the sampling stage, by applying appropriate transformations on the running time observations and then using suitable curve fitting algorithms, it is possible to obtain useful running time predictions and an approximate running time function for the model construction time of a given machine learning algorithm.

Appendix

Appendix A

Proof—NNLS solution vector

The following discussion and proof are extracted and adapted from (Lawson & Hanson, 1974). The NNLS algorithm gives a solution vector for the non-negative least-squares. On termination the solution vector a satisfies

$$a_j > 0, j \in P;$$

$$a_j = 0, j \in Z,$$

and is a solution vector for the least-squares problem

$$X_P a \cong f.$$

The dual vector w satisfies

$$w_j > 0, j \in P;$$

$$w_j = 0, j \in Z,$$

and

$$w = X^T(f - Xa).$$

The above equations constitute the Kuhn-Tucker conditions characterizing a solution vector a for problem NNLS.

Before discussing the convergence of algorithm NNLS it will be convenient to establish the following lemma:

Lemma A1: Let A be an $m \times n$ matrix of rank n and let b be an m-vector satisfying

$$A^T b = \begin{bmatrix} 0 \\ . \\ . \\ . \\ 0 \\ \omega \end{bmatrix} \tag{A.1}$$

with $\omega > 0$.

If $\hat{a}$ is the least-squares solution of $Aa \cong b$, then

$$\hat{a}_n > 0,$$

where $\hat{a}_n$ denotes the nth component of $\hat{a}$.

Proof: Let Q be an $m \times m$ orthogonal matrix that zeros the sub-diagonal elements in the first $n-1$ columns of A, thus

$$Q[A : b] = \begin{bmatrix} R & s & u \\ 0 & t & v \end{bmatrix}, \tag{A.2}$$

where R is upper triangular and nonsingular. Since Q is orthogonal the conditions (A.1) imply

$$R^T u = 0 \tag{A.3}$$

and

$$s^T u + t^T v = \omega > 0. \tag{A.4}$$

Since R is nonsingular, Equation (A.3) implies that $u = 0$. Thus Equation (A.4) reduces to

$$t^T v = \omega > 0.$$

From equation (A.2) it follows that the nth component $\hat{a}_n$ of the solution

vector $\hat{a}$ is the least-squares solution of the reduced problem

$$ta_n \cong v. \tag{A.5}$$

Since the pseudoinverse of the column vector t is ${}^{t^T}/_{(t^T t)}$, the solution of problem (A.5) can be immediately written as

$$\hat{a}_n = \frac{t^T v}{t^T t} = \frac{\omega}{t^T t} > 0,$$

which completes the proof of **Lemma A1**.

Appendix B

A list of 41 WEKA algorithms used for this work

RandomForest

SMO

BFTree

DecisionStump

MultilayerPerceptron

ClassificationViaRegression

ConjunctiveRule

Logistic

J48

LADTree

DecisionTable

J48graft

StackingC

AttributeSelectedClassifier

ClassificationViaClustering

OneR

NNge

RBFNetwork

IB1

PART

ADTree

Stacking

RandomTree

LWL

AdaBoostM1

IBk

Bagging

Vote

BayesNet

FT

DTNB

Ridor

KStar

NBTree

LMT

NaiveBayes

ZeroR

REPTree

SimpleCart

VotedPerceptron

JRip

Appendix C

Data set generator

In this Appendix, parameters used for the data generator are given.

Generator: "weka.datagenerators.classifiers.classification.Agrawal"

Parameter S: 1

Parameter F: 1

Parameter P: 0.05

Header of the data set:

@attribute salary numeric

@attribute commission numeric

@attribute age numeric

@attribute elevel {0,1,2,3,4}

@attribute car {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}

@attribute zipcode {0,1,2,3,4,5,6,7,8}

@attribute hvalue numeric

@attribute hyears numeric

@attribute loan numeric

@attribute group {0,1}

Appendix D

Additional results

This Appendix gives some figures and tables of the experimental results that are not included in the main text.

Setup ID	Training size	PSLR	BC	LADDER	LsF	LsR	LsSeq	LadF	LadR	nnlsF	nnlsR	OneTest
1	5	2%	10%	5%	16%	3%	13%	4%	24%	1%	9%	15%
2	6	4%	7%	3%	4%	4%	6%	12%	34%	1%	6%	19%
3	7	3%	9%	8%	3%	9%	5%	10%	30%	0%	9%	13%
4	8	3%	6%	17%	5%	5%	3%	11%	28%	1%	5%	18%
5	9	3%	5%	10%	9%	13%	0%	13%	18%	4%	4%	19%
6	10	3%	7%	12%	13%	13%	1%	15%	10%	4%	10%	13%
7	11	0%	9%	18%	7%	18%	0%	14%	11%	2%	9%	13%
8	12	3%	5%	18%	6%	10%	1%	14%	11%	8%	9%	16%
9	13	3%	5%	13%	15%	10%	0%	10%	18%	3%	8%	18%
Total		3%	8%	9%	8%	8%	5%	10%	24%	2%	8%	16%

Table D.1: The prediction performance of the 11 estimators over 41 WEKA machine learning algorithms under nine different training/testing setups

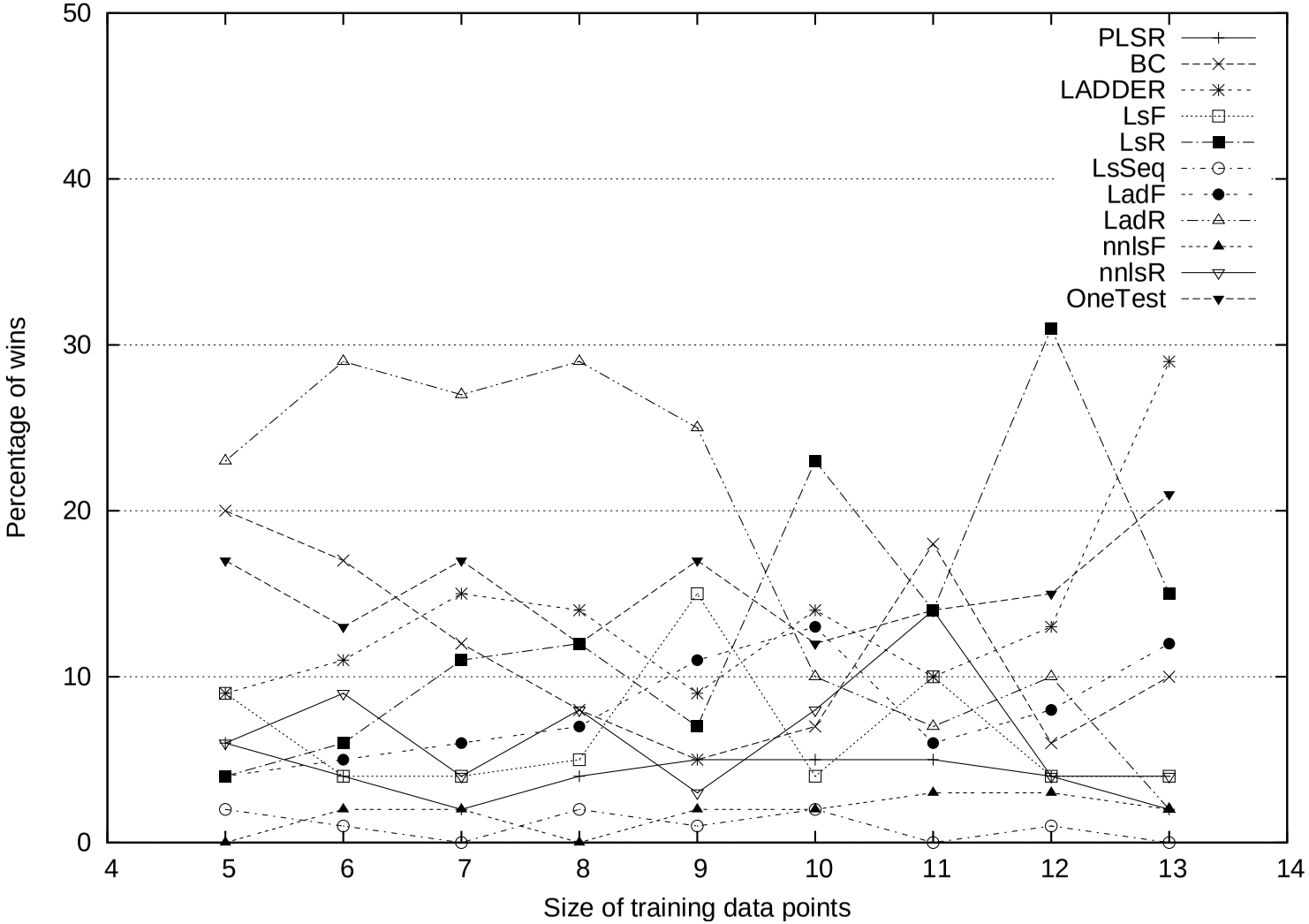


Figure D.1: Prediction performance curves of the 11 estimators as the number of observations increases, evaluated by examining the absolute error, on 41 WEKA classifiers. Running time data were obtained using method B described in Section 4.3

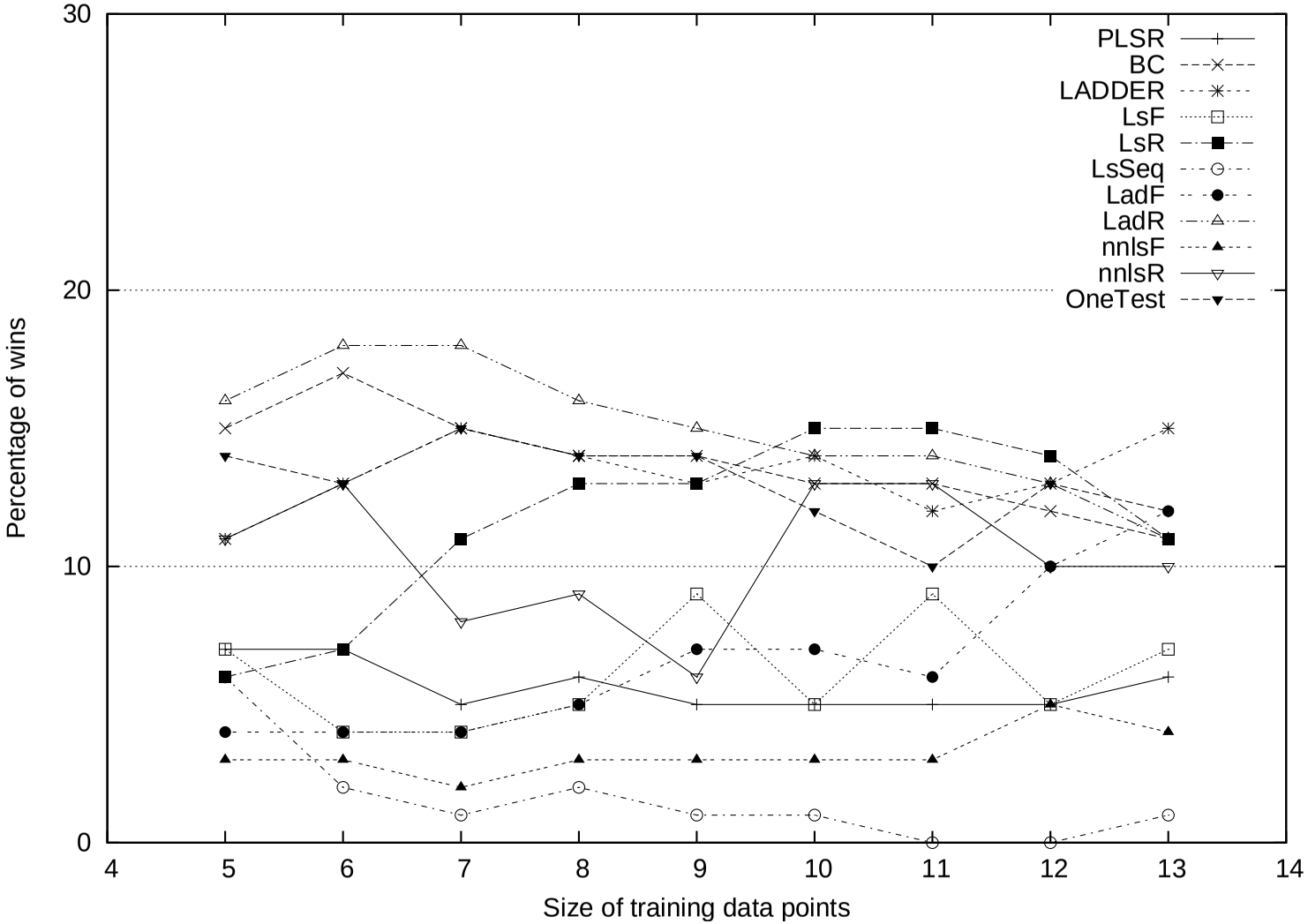


Figure D.2: Prediction performance curves of the 11 estimators as the number of observations increases, evaluated by examining the quality of predictions of these estimators, on 41 WEKA classifiers. Running time data were obtained using method B described in Section 4.3

Rank	Estimator	Percentage of wins
1	LadR	22%
2	OneTest	15%
3	BC	13%
4	LADDER	12%
5	LsR	11%
6	LsF	7%
6	LadF	7%
6	nnlsR	7%
9	PSLR	4%
10	lsSeq	1%
11	nnlsF	1%

Table D.2: A ranked list of the 11 estimators. Ranking positions are based on the prediction performance of the 11 estimators over 41 WEKA machine learning algorithms under nine different training/testing setups, using absolute error as the evaluation criterion. Running time data were obtained using method B described in Section 4.3

Rank	Estimator	Percentage of wins
1	LadR	16%
2	BC	15%
3	OneTest	13%
3	LADDER	13%
5	LsR	11%
6	nnlsR	10%
7	PSLR	6%
7	LsF	6%
7	LadF	6%
10	LsSeq	2%
11	nnlsF	1%

Table D.3: A ranked list of the 11 estimators. Ranking positions are based on the experimental results of the prediction performance of the 11 estimators over 41 WEKA machine learning algorithms under nine different training/testing setups, using absolute errors as the evaluation criterion. Running time data were obtained using method B described in Section 4.3

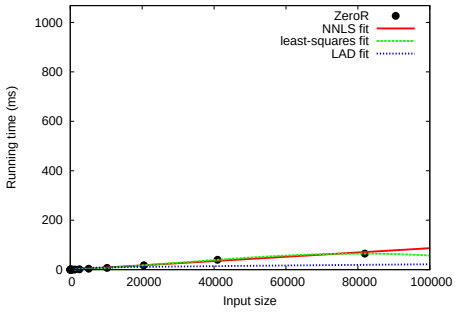
Appendix E

Curve fitting using least-squares, LAD and NNLS

This Appendix gives the curves fitted by using least-squares, LAD and NNLS algorithms on running time data of 41 WEKA machine learning algorithms.

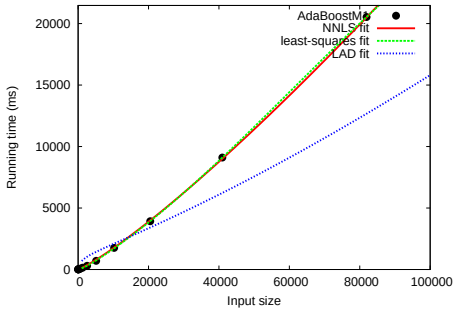
E.1 Curves - running time data measured using method A

The running time data used in the following figures were obtained using the method A described in Section 4.3.

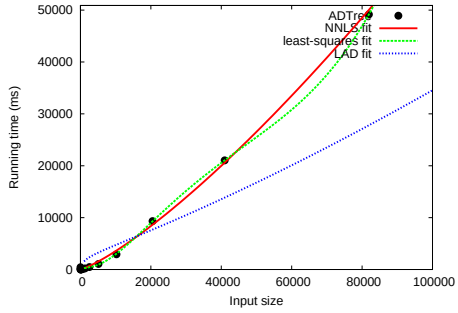


(a) ZeroR

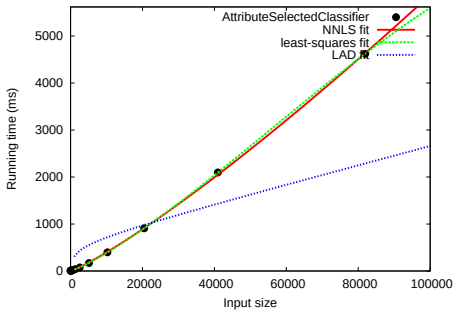
Figure E.1: Running time data curve fitting



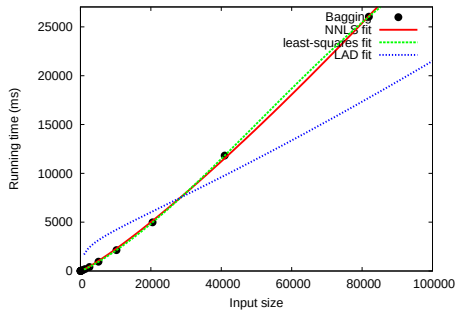
(a) AdaBoostM1



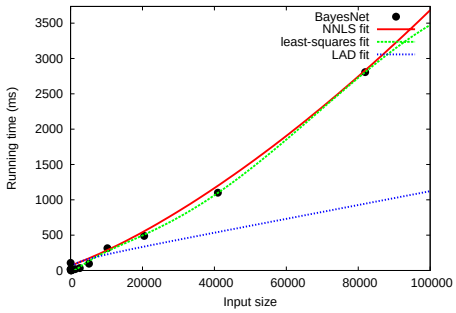
(b) ADTree



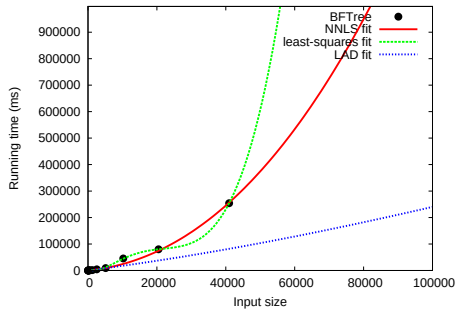
(c) AttributeSelectedClassifier



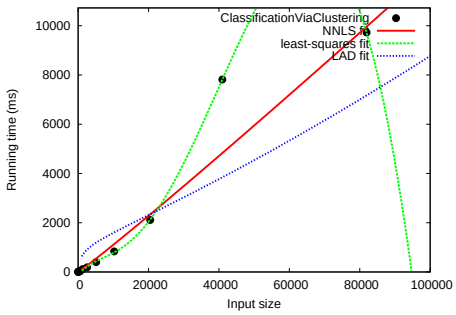
(d) Bagging



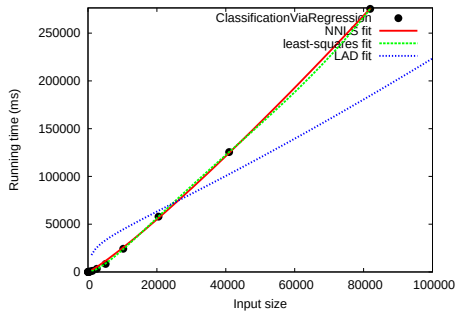
(e) BayesNet



(f) BFTree

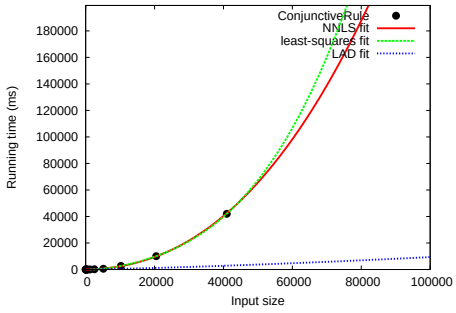


(g) ClassificationViaClustering

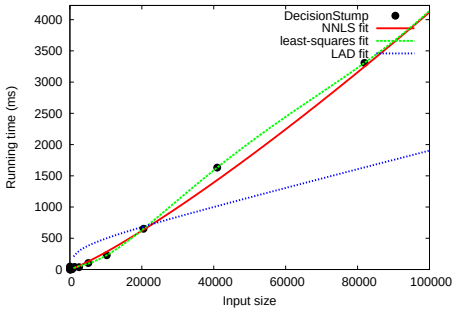


(h) ClassificationViaRegression

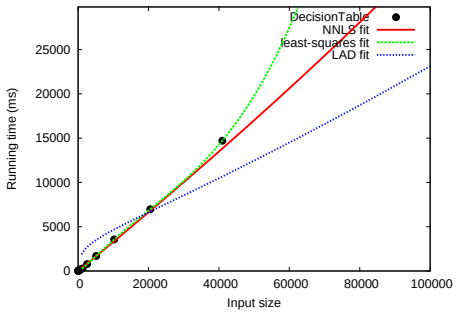
Figure E.2: Running time data curve fitting



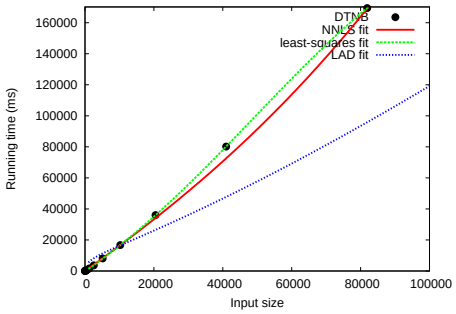
(a) ConjunctiveRule



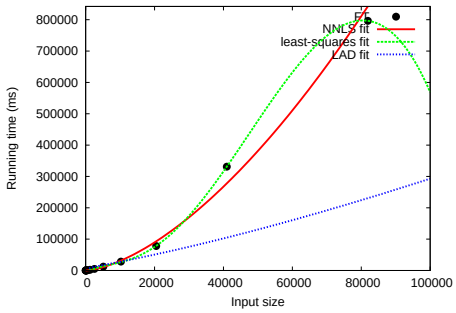
(b) DecisionStump



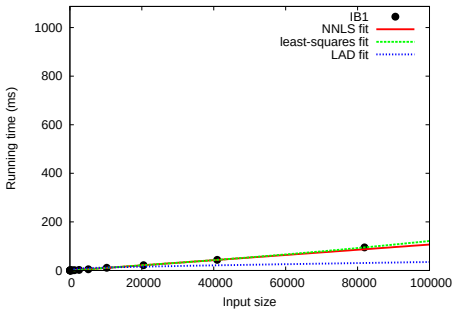
(c) DecisionTable



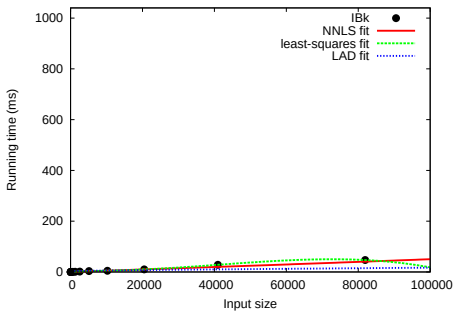
(d) DTNB



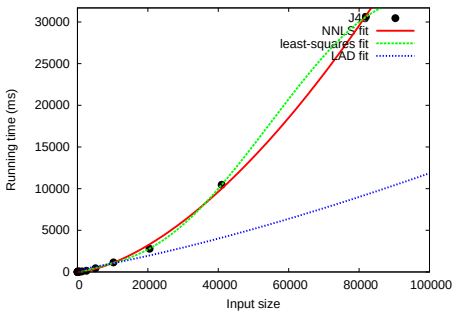
(e) FT



(f) IB1

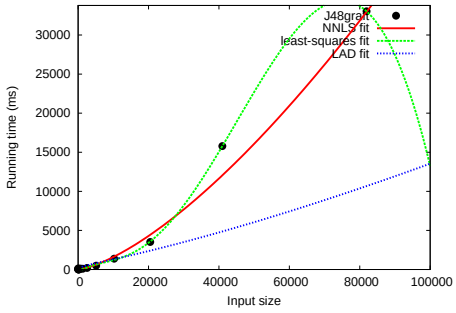


(g) IBk

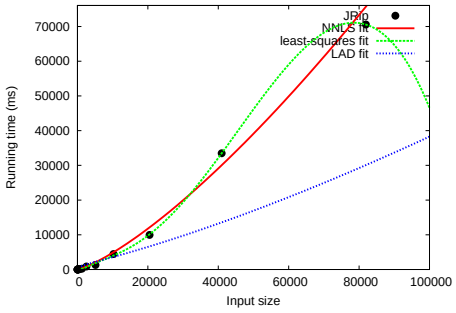


(h) J48

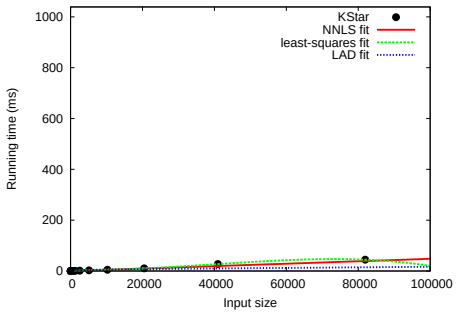
Figure E.3: Running time data curve fitting



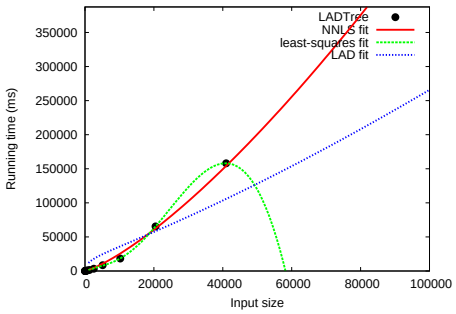
(a) J48graft



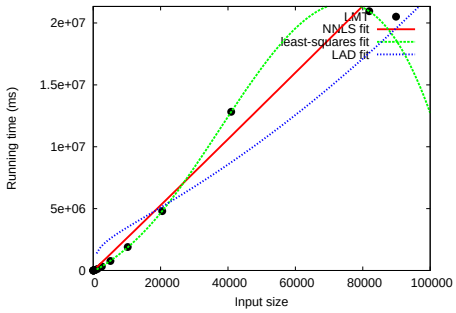
(b) JRip



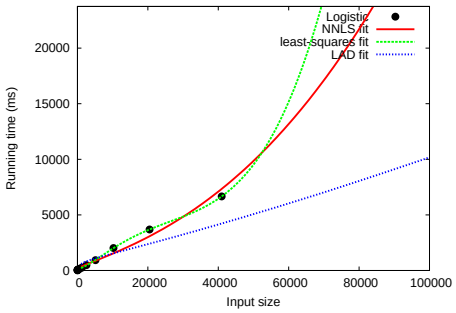
(c) KStar



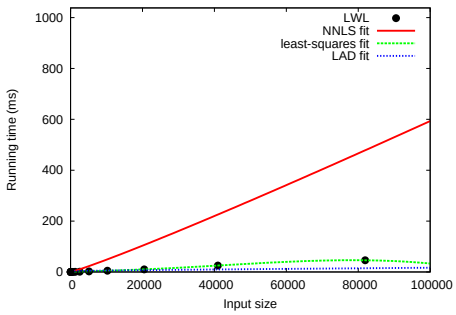
(d) LADTree



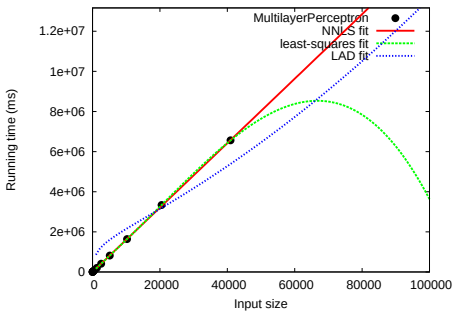
(e) LMT



(f) Logistic

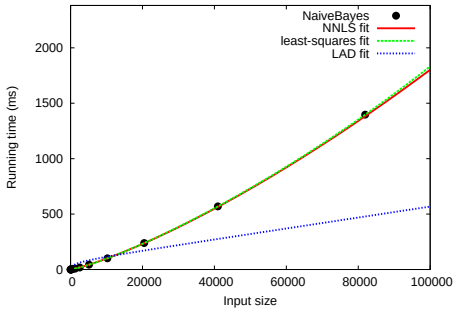


(g) LWL

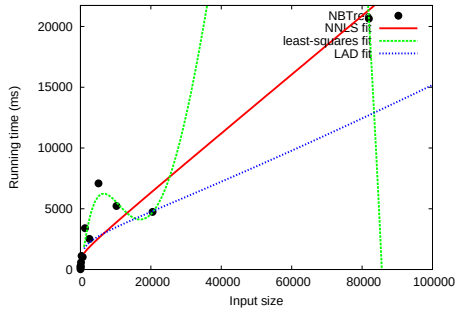


(h) MultilayerPerceptron

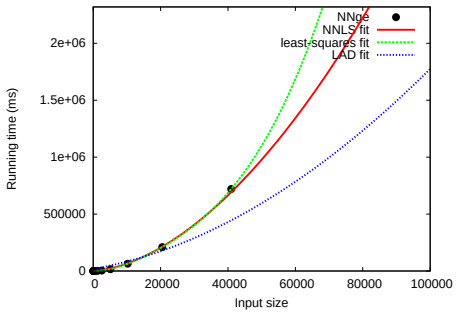
Figure E.4: Running time data curve fitting



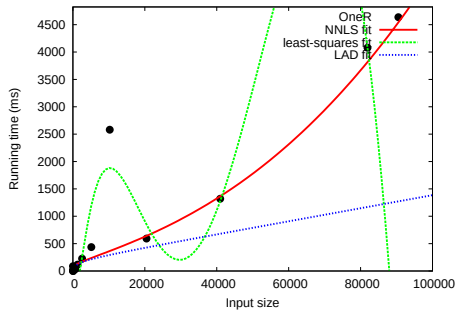
(a) NaiveBayes



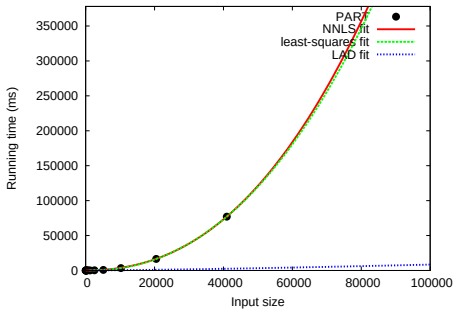
(b) NBTree



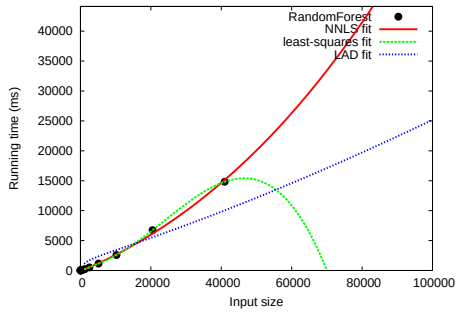
(c) NNge



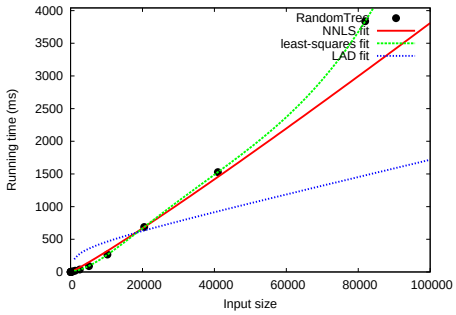
(d) OneR



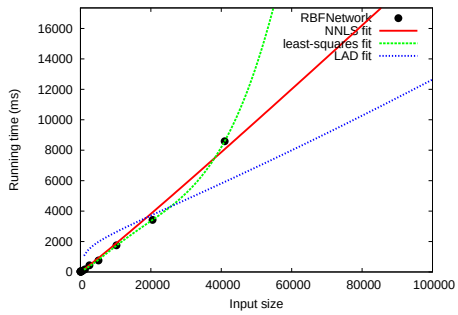
(e) PART



(f) RandomForest

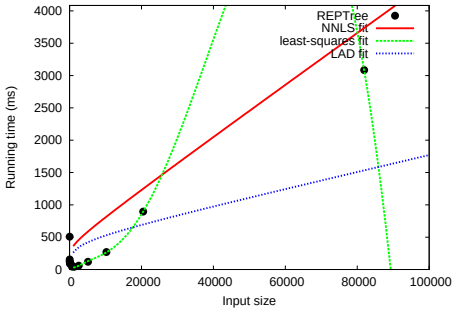


(g) RandomTree

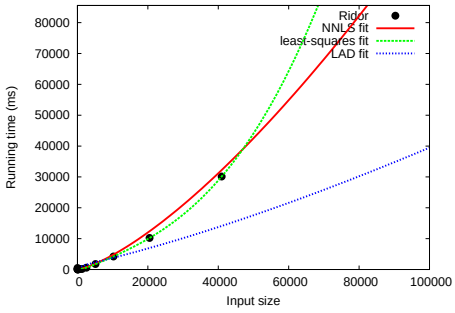


(h) RBFNetwork

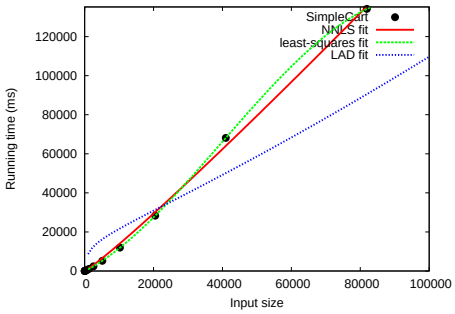
Figure E.5: Running time data curve fitting



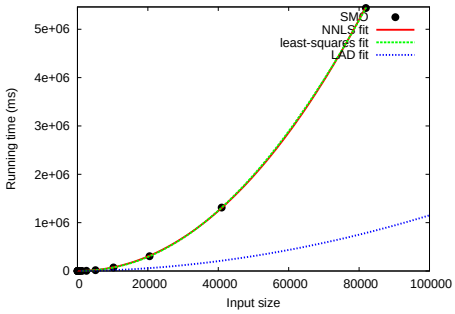
(a) REPTree



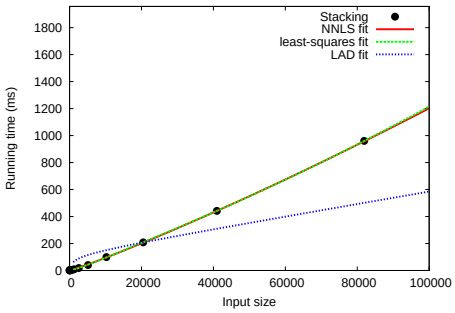
(b) Ridor



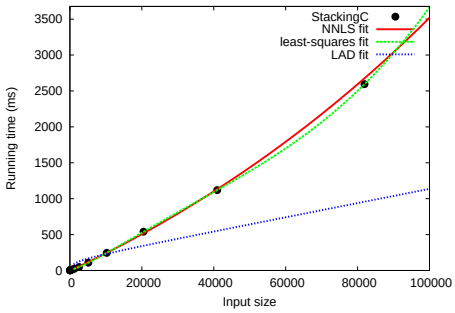
(c) SimpleCart



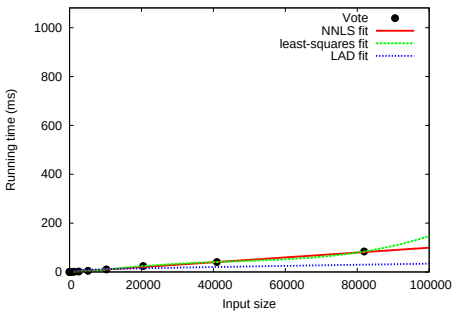
(d) SMO



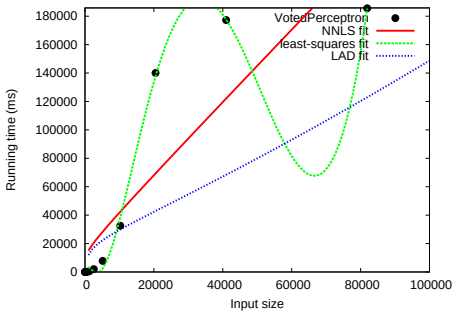
(e) Stacking



(f) StackingC



(g) Vote

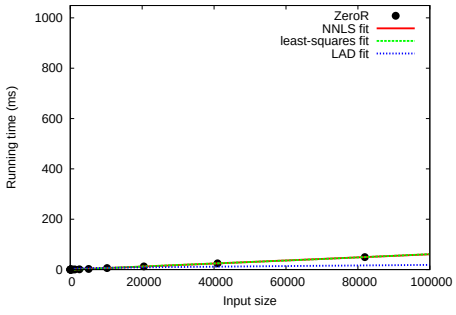


(h) VotedPerceptron

Figure E.6: Running time data curve fitting

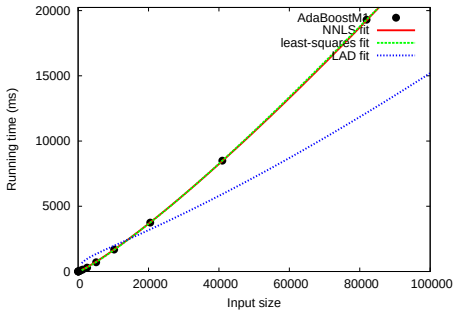
E.2 Curves - running time data measured using method B

The running time data used in the following figures were obtained using the method B described in Section 4.3.

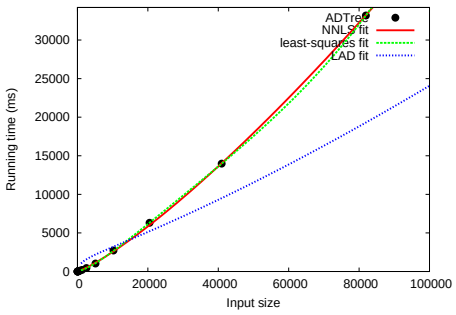


(a) ZeroR

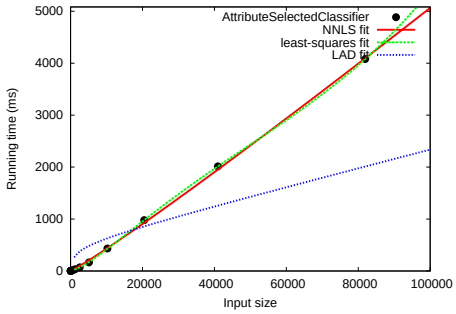
Figure E.7: Running time data curve fitting



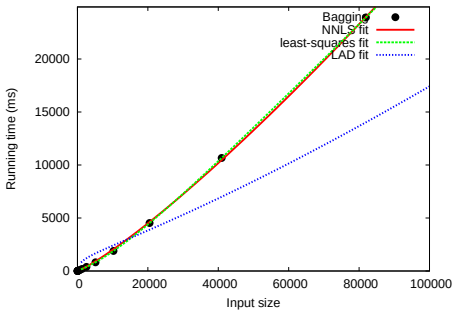
(a) AdaBoostM1



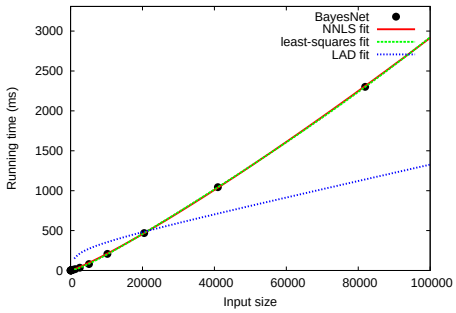
(b) ADTree



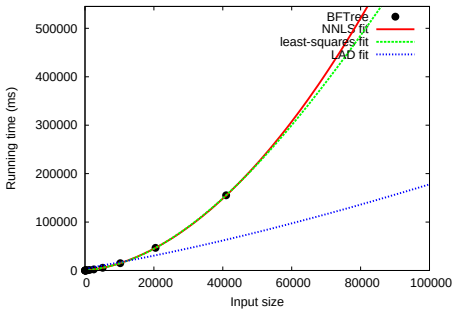
(c) AttributeSelectedClassifier



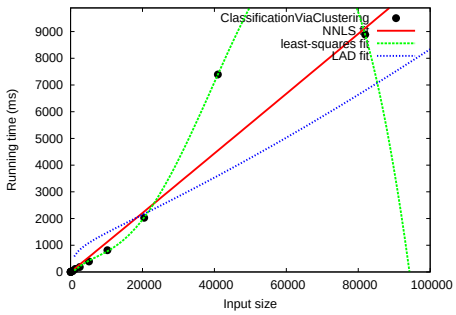
(d) Bagging



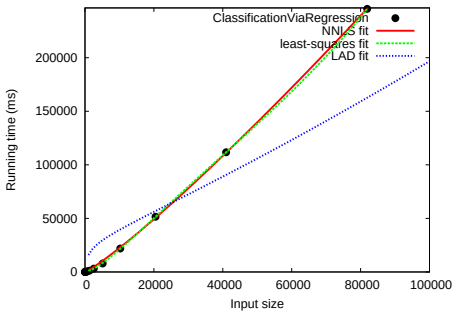
(e) BayesNet



(f) BFTree

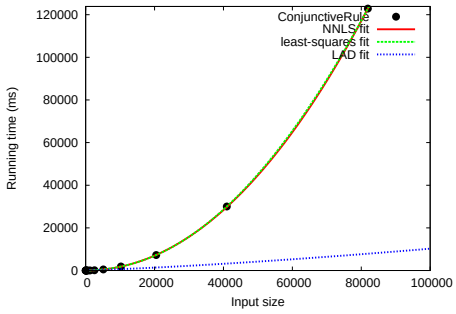


(g) ClassificationViaClustering

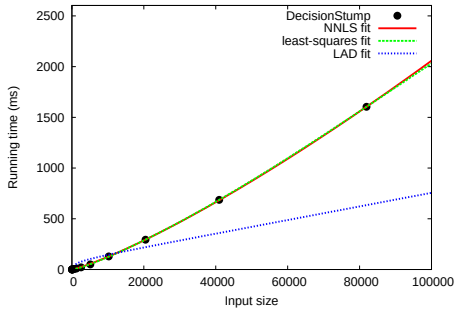


(h) ClassificationViaRegression

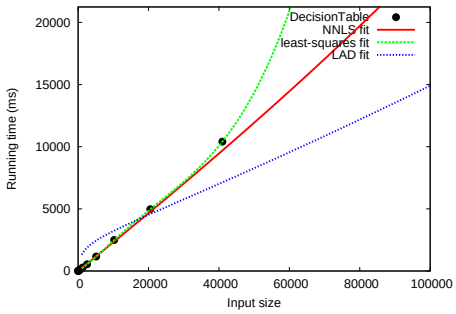
Figure E.8: Running time data curve fitting



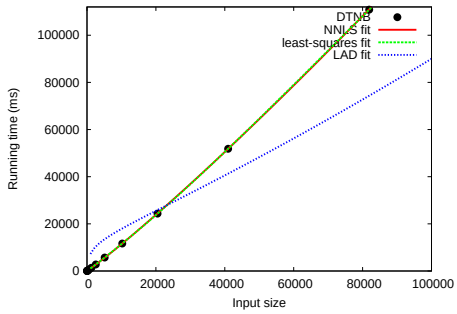
(a) ConjunctiveRule



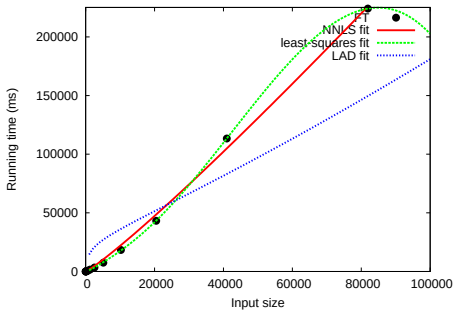
(b) DecisionStump



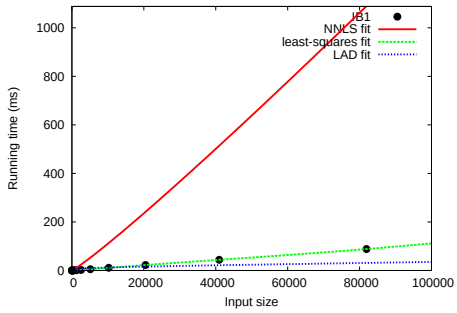
(c) DecisionTable



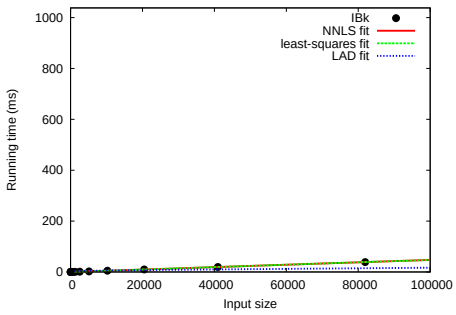
(d) DTNB



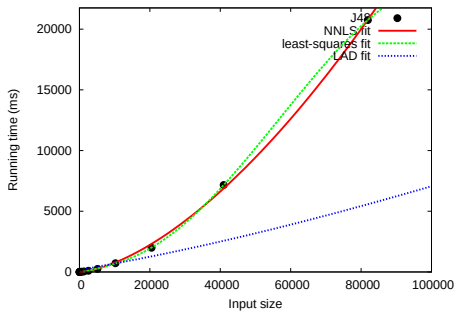
(e) FT



(f) IB1

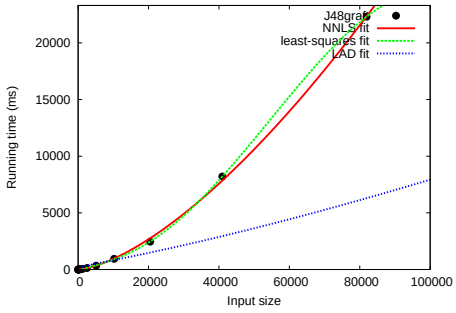


(g) IBk

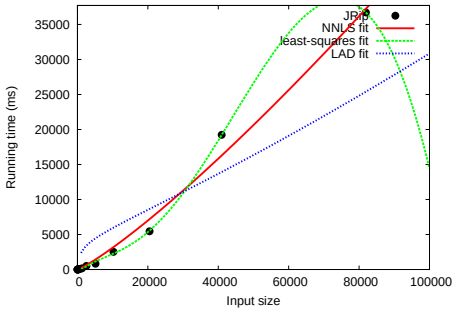


(h) J48

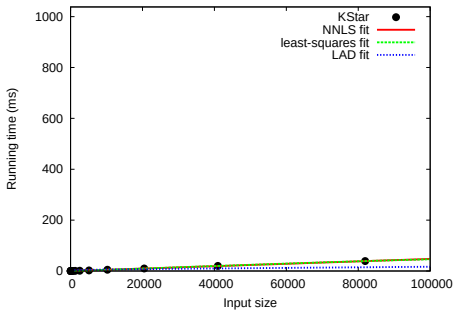
Figure E.9: Running time data curve fitting



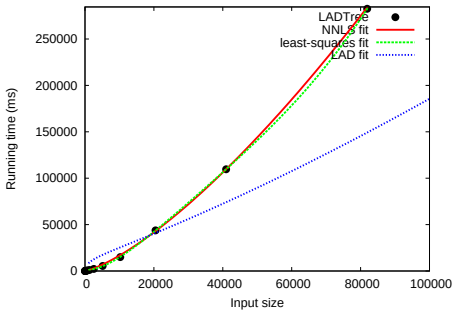
(a) J48graft



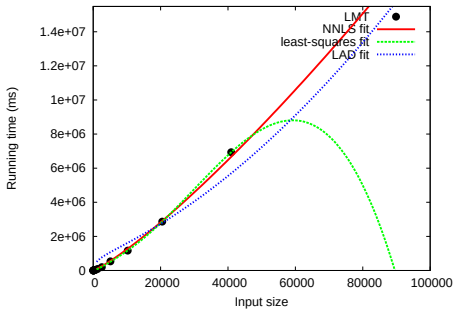
(b) JRip



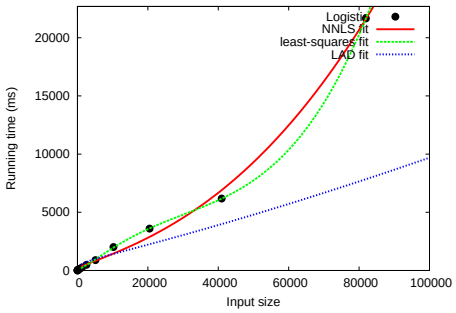
(c) KStar



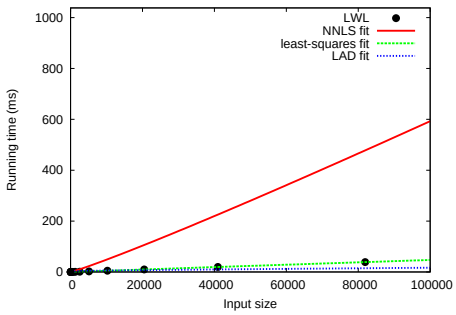
(d) LADTree



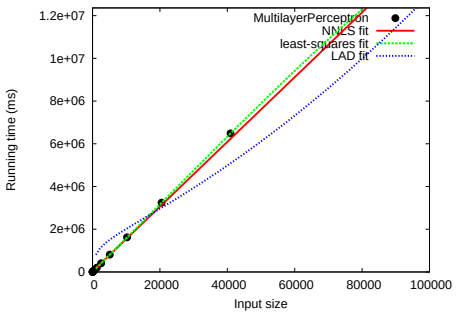
(e) LMT



(f) Logistic

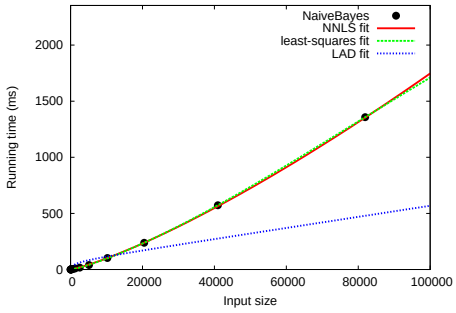


(g) LWL

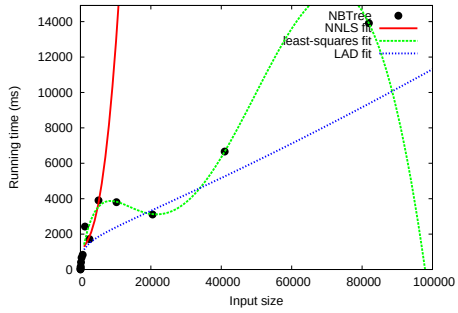


(h) MultilayerPerceptron

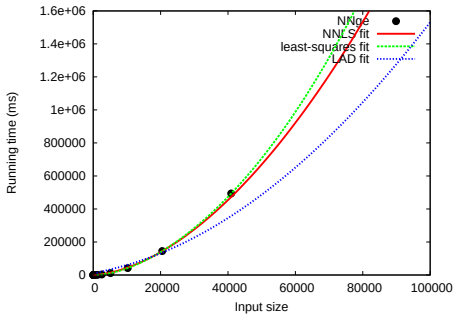
Figure E.10: Running time data curve fitting



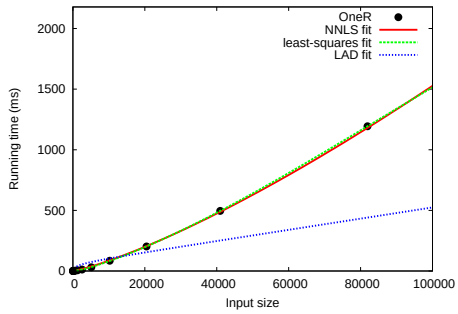
(a) NaiveBayes



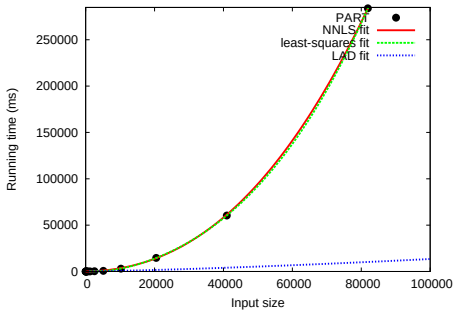
(b) NBTree



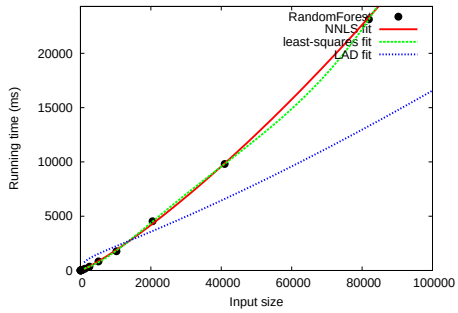
(c) NNge



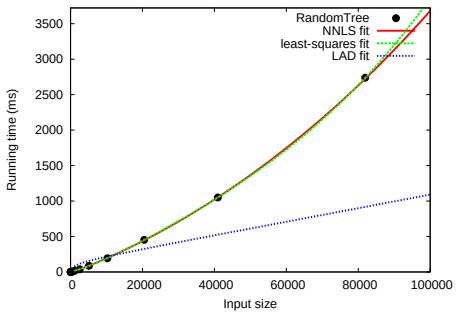
(d) OneR



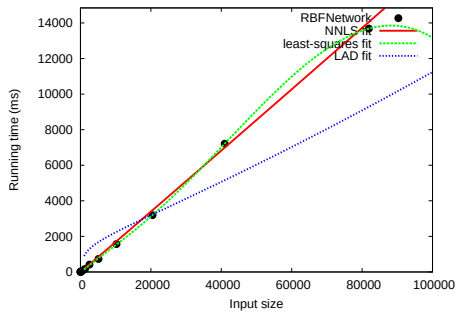
(e) PART



(f) RandomForest

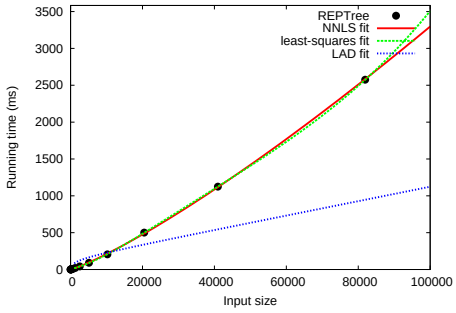


(g) RandomTree

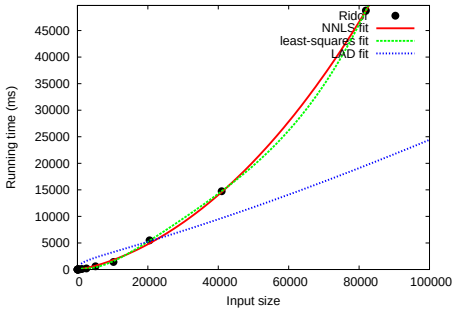


(h) RBFNetwork

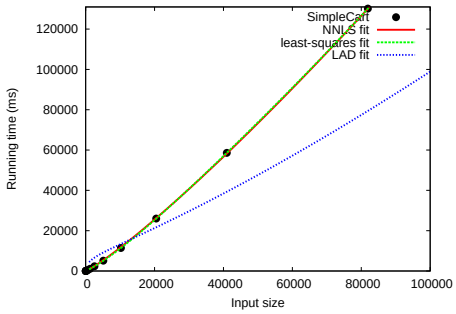
Figure E.11: Running time data curve fitting



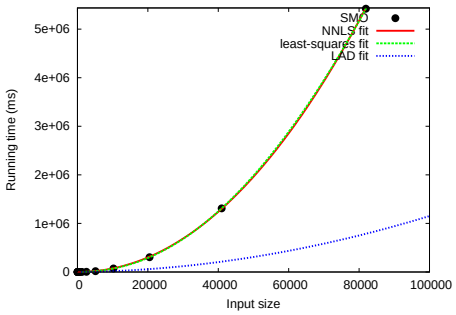
(a) REPTree



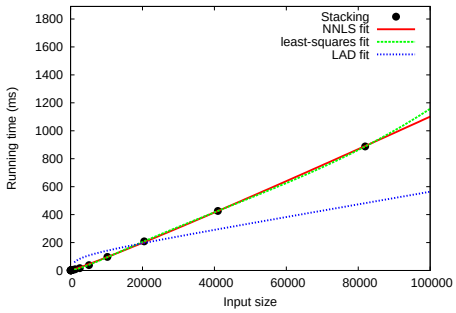
(b) Ridor



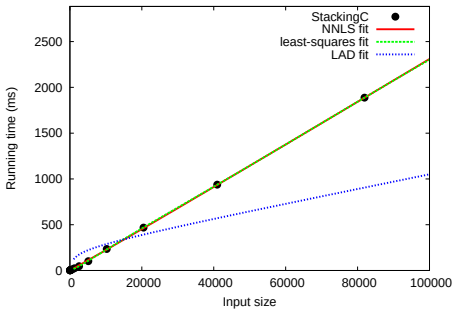
(c) SimpleCart



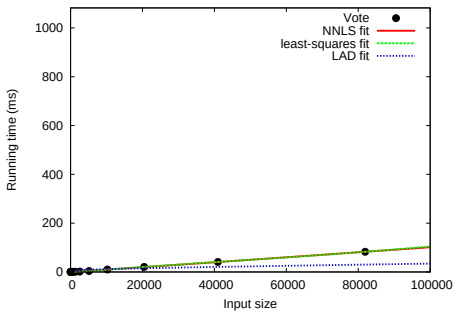
(d) SMO



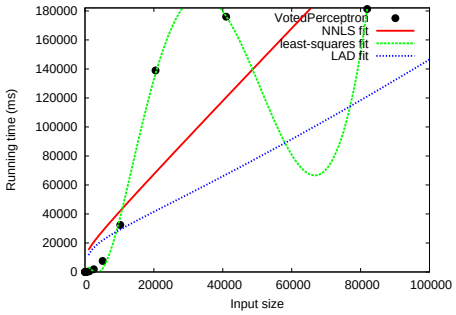
(e) Stacking



(f) StackingC



(g) Vote



(h) VotedPerceptron

Figure E.12: Running time data curve fitting

Bibliography

Akaike, Hirotugu. 1974. *A new look at the statistical model identification.* 6, vol. 19.

Anderson, T. W., & Darling, D. A. 1952. Asymptotic theory of certain goodness-of-fit criteria based on stochastic processes. *Pages 193–212 of: Annals of Mathematical Statistics*, vol. 23.

Asuncion, A., & Newman, D.J. 2007. *UCI Machine Learning Repository.*

Birkes, David, & Dodge, Yadolah. 1993. *Alternative Methods of Regression.* Wiley.

Box, G. P., & Cox, D. R. 1964. An analysis of transformations. *Pages 211–246 of: Journal of the Royal Statistical Society*, vol. 26.

Boyer, Brent. 2008. *Robust Java benchmarking.* Elliptic Group.

Christensen, Ronald. 2001. *Advanced linear modeling.* 2 edn. Springer-Verlag.

Cook, R. D., & Weisberg, S. 1999. *Applied regression including computing and graphics.* John Wiley & Sons.

Elgot, C., & Robinson, A. 1964. Random access stored program machines, an approach to programming languages. *Pages 365–399 of: J. ACM*, vol. 11.

Ellis, Steven P. 1998. Instability of Least Squares, Least Absolute Deviation and Least Median of Squares Linear Regression. *Pages 337–350 of: Statistical Science*, vol. 13.

Fiacco, Anthony V., & McCormick, Garth P. 1968. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* Research Analysis Corporation.

Gill, Philip E., Murray, Walter, & Wright, Margaret H. 1981. *Practical Optimization.* Academic Press.

Goodrich, M. T., & Tamassia, R. 2002. *Algorithm design: Foundations, analysis, and Internet examples.* John Wiley & Sons.

Hall, M. A. 1999. *Correlation-based Feature Subset Selection for Machine Learning.* Ph.D. thesis, The Univeristy of Waikato.

Kohavi, Ron, & John, George H. 1996. *Wrappers for Feature Subset Selection.*

Kohavi, Ron, & Sommerfield, Dan. 1995. *Feature Subset Selection Using the Wrapper Method: Overfitting and Dynamic Search Space Topology.*

Last, Mark. 2009. Improving data mining utility with projective sampling. *Pages 487–496 of: KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining.* New York, NY, USA: ACM.

Lawson, Charles L., & Hanson, Richard J. 1974. *Solving Least Squares Problems.* Prentice-Hall.

Lehmann, E.L., & Casella, George. 1998. *Theory of Point Estimation.* 2 edn. Springer.

Li, Jianliang, Jiang, Yong, & Wang, Guangxian. 2000. *Numerical methods (Jisuanji Shuzhi Fangfa).* Dong Nan University Press.

McGeoch, Catherine, Sanders, Peter, Fleischer, Rudolf, Cohen, Paul R., & Precup, Doina. 2002. Using finite experiments to study asymptotic performance. 93–126.

Mitchell, Tom. 1997. *Machine Learning.* McGraw Hill.

Moore, David S., & McCabe, George P. 1999. *Introduction to the practice of statistics.* 3 edn. W. H. Freeman and Company.

Mosteller, F., & Tukey, J. W. 1977. *Data analysis and regression: A second course in statistics.* Addison-Wesley.

Nocedal, Jorge, & Wright, Stephen J. 1999. *Numerical Optimization.* Springer.

Papoulis, A. 1984. *Probability, Random Variables, and Stochastic Processes.* 2 edn. McGraw-Hill.

Spiegel, M. R. 1992. *Theory and Problems of Probability and Statistics.* McGraw-Hill.

Spiegel, Murray R., & Stephens, Larry J. 2008. *Theory and problems of Statistics.* 4 edn. McGraw-Hill.

Sun, Quan. 2008 (7). *Sampling-based running time prediction.* Dissertation.

Trochim, W. 2006. *Deduction and induction.* World Wide Web electronic publication.

Tukey, J. W. 1977. *Exploratory data analysis.* Addison-Wesley.

Vilalta, Ricardo, & Drissi, Youssef. 2002. A perspective view and survey of meta-learning. *Pages 77–95 of: Artificial Intelligence Review*, vol. 18. Kluwer Academic.

Wang, Yong. 2000. *A New Approach to Fitting Linear Models in High Dimensional Spaces.* Ph.D. thesis, The University of Waikato.

Witten, Ian H., & Frank, Eibe. 2005. *Data Mining: Practical Machine Learning Tools and Techniques.* 2 edn. San Francisco, CA: Morgan Kaufmann.