THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Parameter Tuning Using Gaussian Processes

Jinjin Ma

This thesis is submitted in partial fulfillment of the requirements for the Degree of Master of Computer Science at the University of Waikato.

January 2012

Abstract

Most machine learning algorithms require us to set up their parameter values before applying these algorithms to solve problems. Appropriate parameter settings will bring good performance while inappropriate parameter settings generally result in poor modelling. Hence, it is necessary to acquire the "best" parameter values for a particular algorithm before building the model. The "best" model not only reflects the "real" function and is well fitted to existing points, but also gives good performance when making predictions for new points with previously unseen values.

A number of methods exist that have been proposed to optimize parameter values. The basic idea of all such methods is a trial-and-error process whereas the work presented in this thesis employs Gaussian process (GP) regression to optimize the parameter values of a given machine learning algorithm. In this thesis, we consider the optimization of only two-parameter learning algorithms. All the possible parameter values are specified in a 2-dimensional grid in this work. To avoid brute-force search, Gaussian Process Optimization (GPO) makes use of "expected improvement" to pick useful points rather than validating every point of the grid step by step. The point with the highest expected improvement is evaluated using cross-validation and the resulting data point is added to the training set for the Gaussian process model. This process is repeated until a stopping criterion is met. The final model is built using the learning algorithm based on the best parameter values identified in this process.

In order to test the effectiveness of this optimization method on regression and classification problems, we use it to optimize parameters of some well-known machine learning algorithms, such as decision tree learning, support vector machines and boosting with trees. Through the analysis of experimental results

obtained on datasets from the UCI repository, we find that the GPO algorithm yields competitive performance compared with a brute-force approach, while exhibiting a distinct advantage in terms of training time and number of cross-validation runs. Overall, the GPO method is a promising method for the optimization of parameter values in machine learning.

Acknowledgements

It is a great pleasure to express my gratitude to the many people who made this thesis possible.

First, I would like to thank my supervisors Associate Professor Eibe Frank and Professor Geoff Holmes for supervising the thesis. They suggested many new research ideas and explained things very clearly and simply with great effort and enthusiasm. They provided a lot of help when I met different problems. During the thesis-writing period, they always gave me much encouragement and sound advice as well as good proof-reading. I also would like to thank Quan Sun and John Moriarty for proofreading the draft of the thesis.

Further thanks go to my colleagues in the machine learning group. They are a group of lovely people who always help each other. I enjoyed discussing problems and sharing ideas with them. They provided such a friendly and excellent research environment.

I wish to thank my friends for bringing me joy during the hard times. They encouraged me, supported me and helped me all the time.

I also thank Clint Dilks for giving me technical support and maintenance for my important experiments. A special thanks goes to the computer servers I used for my experiments. They might be tired out if they keep running. They may have spent a longer time than I did on this thesis.

I am very grateful to the scholarship from the Machine Learning Group. Many thanks for the financial support during the study. I can not imagine doing it without this huge help.

Lastly, and most importantly, I am forever indebted to my parents far away from here. I thank them for giving every support and endless love when it was most required. I dedicate this thesis to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years machine learning algorithms have been applied to various areas; optimizing the model they produce is becoming a very important topic. Apart from a few simple algorithms that do not have tuning parameters, most machine learning algorithms may suffer from inappropriate parameter settings because the parameter values must be set before applying algorithms to solve problems [15], even though default values in some algorithms perform satisfactorily.

Generally, there is no clear and unique method to find the "best" parameter settings, and a number of approaches have been proposed that try to search for "good" parameter values. However, some of these approaches may easily fall into a suboptimal situation [30], and others have to do a brute-force search, such as Grid Search. Grid Search is an exhaustive approach without any optimization during the search process, starting from the minimum point of a grid of parameter values specified by the user to the maximum point. It has the obvious advantage that every point in the grid is evaluated before the "best" point is chosen. Assuming the grid is chosen appropriately, it guarantees that the "best" point is close to the globally optimal parameters rather than in a suboptimal area. The aim of this thesis is to achieve the same high-quality parameter settings as Grid Search using a speedier process by applying Gaussian process regression to model performance depending on parameter settings.

There are prior works on parameter optimization methods for machine

learning algorithms. An idea based on principles of the design of experiments (DOE) was presented in [43] that can identify the meta parameter settings for support vector machines. This algorithm can successfully find optimal or near-optimal parameters and has already been applied to several problems in HP Laboratories. Lin [30] proposed a novel scatter search-based approach (SS + DT) to acquire parameter settings and demonstrated that the experimental results using this approach for the C4.5 algorithm are better than those obtained by other approaches. Genetic algorithms have also been applied as a parameter optimization tool for finding maximum accuracy predictive models. However, this kind of algorithm can not guarantee that the point found is near the optimal solution due to the inherent limitations of the heuristic search approach [44]. Another approach is to learn a model in parameter space based on on-line Gaussian process models. In [13], that approach is applied to the optimization of support vector machines; and experimental results on a small collection of benchmark datasets indicate that it is effective. In this thesis, the same basic approach is evaluated more extensively using a great number of datasets and a diverse set of learning algorithms.

1.1 Objectives of Parameter Optimization Using Gaussian Process Models

The goal of parameter optimization in machine learning is to build a model using appropriate parameter settings to get the best predictive performance, such as accuracy in a classification problem, or a particular correlation coefficient in a regression problem. Applying a Gaussian process model to optimize parameter values is a promising optimization methodology in terms of the cost of computation and the effectiveness of the model. Gaussian Process Optimization makes use of a Gaussian process model, trained on performance scores collected already, to obtain predictive distributions in order to estimate performance scores for new candidate points, which then decides suitable parameter settings from these. The basic approach for picking high-quality candidate points (i.e. parameter combinations) is to employ the so-called "expected improvement" for each point [20], which is calculated based on the Gaussian

process model.

In this optimization context, the initial Gaussian process model learns through a few data points corresponding to initial performance scores treated as a training set. The next sample is selected by maximizing the expected improvement with respect to this initial Gaussian process model amongst all candidate data points in the search space. The new sample, paired with its performance score, is estimated using methods such as cross-validation, and then is added to the training set to update all the parameters of the Gaussian process model. Each new point improves prediction accuracy and reduces uncertainty to some extent. This process is applied iteratively until a stopping criterion is met. Finally, the model, using optimized parameter values encountered in the optimization process, can be applied to predict unknown class labels in the classification case, or target values in the regression case.

This optimization methodology exploits the merits of Gaussian process learning: not only can it obtain a full predictive distribution for any point in space, thus making it possible to calculate expected improvement, but also the properties of the known samples can be learned easily and flexibly using a so-called "kernel function". The natural characteristics of Gaussian process models make the optimization solution simple.

In this thesis, we will consider this methodology to tune the parameters of various machine learning algorithms, both in classification and regression problems. For most machine learning algorithms, two parameters or even multiple parameters contribute to the functions of the model, even though in some special cases there is no parameter to tune, e.g. in the case of a decision stump classifier. We will discuss how optimization using the Gaussian process model can be performed effectively and cheaply. As we will see, maximizing expected improvement using Gaussian process model balances global and local searches successfully and yields good parameter settings in an efficient manner.

1.2 Structure of the Thesis

The remainder of this thesis is organized into four chapters as follows. Chapter 2 gives a theoretical explanation of Gaussian process learning with

relevant derivations of equations and then provides detailed background on some well-known machine learning algorithms as the basis for subsequent chapters. It also discusses cross-validation, which is used to evaluate the performance of the generated model.

Chapter 3 briefly introduces Grid Search and Random Search as baseline search methods to be compared to the Gaussian process search method. This chapter also describes the concept of expected improvement and how it works in Gaussian process models. It focuses on the specific procedure of Gaussian Process Optimization and also provides a preliminary view of the performance of optimization based on Gaussian processes. We will see that the selection of Gaussian process parameter values can influence the outcome of the predictive model. Three-dimensional plots of the specified Gaussian process parameter values and the corresponding predictions help our understanding of the effect of these parameters for a particular dataset. As we will see, it is noticeable that the appropriate Gaussian process parameter values, including kernel function values, are an important factor when building models for optimization. We will provide an investigation of this phenomenon in this chapter. This chapter also investigates a stopping criterion to show when the optimization process can reliably stop without influencing the final predictive performance.

Chapter 4 provides a substantial amount of evidence, obtained using various learning algorithms, to prove the effectiveness of Gaussian Process Optimization in both regression and classification learning domains. The experiments are performed on several classical datasets and compare the performance of Gaussian Process Optimization with Grid Search and Random Search according to several measures. Through the comprehensive analysis of these results, we find that Gaussian Process Optimization generally has equivalent performance in terms of the accuracy (classification) and the correlation coefficient (regression), and a distinct advantage in terms of runtime and number of cross-validation runs compared with the other two methods.

Chapter 5 concludes this thesis and finds that Gaussian Process Optimization can be applied successfully to optimize the parameters of machine learning algorithms. Due to the high-quality parameter values chosen by this process, it becomes easier and faster to build a good model. Nevertheless, there is scope

for future work. The optimization methodology, as investigated in this thesis, is limited to two parameters of the specified base algorithms. It would be useful to investigate optimizing multiple parameters. In addition, because of the computational complexity involved, the optimization process still consumes a substantial amount of time on larger datasets. We should investigate further modifications and perform research in the future to make the process even more efficient.

Chapter 2

Background

This thesis is about parameter optimization in machine learning. In this chapter, we review the learning algorithms that are used later in the parameter optimization experiments. These learning algorithm typically have two interacting parameters where optimal performance is obtained when a good pair of values has been determined. They cover a wide range of approaches: regression and classification, decision tree learning, kernel-based methods, and ensemble classifiers. More specifically, Section 2.1 discusses Gaussian processes for regression, Section 2.2 covers decision tree learning, Section 2.3 considers ensemble learning using boosting and Section 2.4 explains the basics of support vector machines. Section 2.5 discusses how cross-validation is used to estimate performance. Section 2.6 contains a brief summary of this chapter.

2.1 Gaussian Processes

Suppose we are given a training dataset D of N data points X_N with inputs x and outputs (or targets) y_N. The inputs x may contain just one variable or may represent an I dimensional vector, i.e. in general there may be many input variables (or attributes). For example, the "CPU" dataset from the UCI repository has 6 input attributes, such as minimum memory and maximum memory; and for the "glass" dataset, there are 10 input elements to discriminate different kinds of glass. The targets y_N are composed of a list of scalars, which are continuous in the regression case, or discrete in the classification case because

there they belong to categorical objects. We consider the regression case only in this section.

In the machine learning context, it is important to discover the relationships between inputs and outputs of a given dataset so that the relationship or the function $f(x)$ built from the observed data points can be used to predict future unknown values for new test data. Generally, there are two common approaches to deal with the predictions [4]. One approach is to fix a certain function according to the characteristic of the given dataset and the experience of the analyst. The predictions of this approach may be accurate and yield good performance if the target is modelled well by the selected function, or the dataset is simple enough for such a function, but this approach may also easily run into the danger of over-fitting, where it can obtain good predictions for the training data, but perform badly when doing test predictions. The other approach is to "give a prior probability to every possible function" [4], which means every possible function will have a chance to become the function chosen afterwards, and the preferred function is more likely to have higher probability.

The inference of the unknown function denoted as $f(x)$ in the latter approach is expressed by the posterior probability, which is based on Bayes rule as given in the following expression [32]:

$$P(f(x)|y_N, X_N) \quad = \quad \frac{P(y_N|f(x), X_N)P(f(x))}{P(y_N|X_N)} \tag{2.1}$$

where the first probability in the numerator on the right hand side, $P(y_N|f(x), X_N)$, is the probability of outputs y_N given inputs X_N on the basis of the function $f(x)$, and the second distribution, $P(f(x))$, is the prior distribution assumed by this model over functions.

For the sake of gaining a better understanding of the principles of Gaussian process regression, let us start with the simple multiple linear regression model $f(x; w)$ with additive noise ε. The model can be stated as follows [4]:

$$f(x) = f(x; w) = x^T w \tag{2.2}$$

$$y = f(x; w) + \varepsilon \tag{2.3}$$

In Equation 2.2, $x^T = \{x_1, x_2, \ldots, x_i\}$ and $w = \{w_1, w_2, \ldots, w_i\}$, where i is the ith index of the inputs and w_i is the coefficient corresponding to this

input. The noise model ε is normally considered to be independently, identically distributed noise added to the function.

The objective of building the mapping function $f(x)$ is not to optimize the performance for the training data points, but to predict well the unknown values for new data points. Ideally, the "real" underlying target function is found. However, this is a question worthy of consideration in the second approach: if there are uncountably many functions to use, it will be impossible to obtain the real function through Equation 2.1. The idea of Gaussian process regression is to omit unnecessary functions from the possible functions and infer the most likely function in the process. More precisely, we try to make a suitable function fit the observed data points (or be "close" to them), and get the full predictions for the infinite objectives on the basis of the properties of the existing finite objectives [4] [32]. Mackay [32] mentions that many popular non-linear modeling methods, such as Bayesian models, neural networks and other related models, e.g. support vector machines, are related to Gaussian process regression under the mathematical viewpoint.

A Gaussian process, as a representative of a stochastic process, is a principled process to learn the characteristics of previously collected samples and to give predictions of likely values of y for future inputs x. The method is frequently used in real-world supervised learning applications of machine learning [2] [18] [17]. The key point for this process is that it can be thought of as a generalization of a Gaussian distribution, as the predictive distribution for future samples is obtained easily and the general properties of the model can be learned from previous samples relatively easily. The model has low complexity of inference when making predictions for future values of x.

According to the definition in a well-known textbook on Gaussian processes [4], a Gaussian process is "a collection of random variables, any finite number of which have a joint Gaussian distribution". A Gaussian distribution is typically specified by its mean and covariance matrix. Similarly, from the view of function-space, a Gaussian process is fully specified by a mean $m(x)$ and covariance function $k(x, x')$ which declares the covariance of the function $f(\cdot)$ at the points x and x' [4]. For each sample x, $f(x)$ is treated as a single Gaussian distribution. Hence, a Gaussian process is aimed at functions while a Gaussian

distribution is defined over vectors. The covariance function is also called the kernel function, formalized as $k(\cdot,\cdot)$. We will use this term later. Following [4], we write the Gaussian process (GP) as

$$f(x) \sim GP(m(x), k(x, x')) \tag{2.4}$$

The two functions $m(x)$ and $k(x, x')$ are:

$$m(x) \quad = \quad GP[f(x)] \tag{2.5}$$

$$k(x, x') \quad = \quad GP[(f(x) - m(x))(f(x') - m(x'))] \tag{2.6}$$

The kernel function $k(x, x')$ between the pair of random variables x and x' is used as a prior over functions and the covariance between the output targets is expressed as a function of the input variables: $k(x, x') = cov(f(x), f(x'))$. The above expression implies that the connection between the functions at an existing point and a new point depends on the relationship of these points. They are likely to have similar target values when x' is close to x, while a point x' far away from x will show a distinct target result normally [4].

There are many kernel functions to use, such as the linear kernel, polynomial kernel and spline kernel [16]. In this thesis we will focus on the Radial Basis Function (RBF) kernel function (or Gaussian kernel function):

$$k(x_i, x_j) \quad = \quad exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$$

It can be re-defined in terms of γ as

$$k(x_i, x_j) \quad = \quad exp(-\gamma||x_i - x_j||^2),$$

where $\gamma = \frac{1}{2\sigma^2}$ and σ is the kernel width parameter. In this case, the output of the kernel depends on the Euclidean distance of x_i from x_j with a certain parameter γ, one of which will be the training data point and the other of which can be regarded as the testing data point. A large σ (or small γ) implies a very smooth fit, with a large neighbourhood of influence for a given data point, which makes it possible to avoid reproducing noise in the samples and to avoid overfitting the model to the data samples used in the training procedure. Similarly, small σ values (large γ) mean the parameter influence on the neighbourhood is much restricted and they give steeper functions.

If users are highly focused on the results on the training data, that is, when a large value of γ is given, it is natural that the results are better because that allows the training procedure to adapt almost perfectly to the data given to it. However, if the model is too adapted and fit to the training samples, the prediction for unseen data samples will not be good. Therefore, deciding on an appropriate value of the γ parameter in the kernel function is a crucial step to get a good balance.

In the above discussion, the Gaussian process is used to define a prior distribution over functions. The prior distribution is not dependent on the training data, but reflects the properties of natural features of the function. Now we will turn to how to make predictions with a Gaussian process. The key assumption is that the posterior predictive distribution $P(y^*|D)$ is Gaussian, corresponding to a certain test point x^*, so that the predictive distribution for an unseen target value $y*$ can be treated as $y^* \sim \mathcal{N}(m(x^*), C(x^*))$, for which $m(x^*)$ and $C(x^*)$ can be computed from the covariance matrix and the observed output values y.

We consider additive noise that is added to the function $f(\cdot)$ in real-world modelling situations as $y = f(x) + \delta$, where δ is a noise parameter in the Gaussian process model. The δ is assumed to be Gaussian noise with mean zero and variance σ^2, which is distributed as $\delta \sim \mathcal{N}(0, \sigma^2)$. So, the prior distribution over x_i and x_j becomes

$$cov(f(x_i), f(x_j)) \quad = \quad k(x_i, x_j) + \sigma^2 \delta_{ij},$$

where δ_{ij} is 1 when $i = j$ and 0 otherwise. If we purely use the observed data points X paired with their targets y, the equation can also be written as

$$cov(y) \quad = \quad K(X, X) + \sigma^2 I$$

.

For some test point x^*, we denote $k(x^*) = k_*$ as the vector of covariances between the test point and the observed N training data points. The mean function $m(x^*)$ and the covariance function $C(x^*)$ with noise parameter σ for this test point can be defined as follows [4]:

$$m(x^*) \quad = \quad k_*^T (K(X, X) + \sigma^2 I)^{-1} y \tag{2.7}$$

$$C(x^*) \quad = \quad k(x_*, x_*) - k_*^T (K(X, X) + \sigma^2 I)^{-1} k_* \tag{2.8}$$

The mean prediction can also be written in another way as a linear combination of n kernel functions through

$$m(x^*) \quad = \quad \sum_{i=1}^{n} \alpha_i k(x_i, x^*)$$

where $\alpha = (K(X, X) + \sigma^2 I)^{-1} y$. From the equations above, we can see that the mean prediction at the test point x^* is a linear combination of output values y while the variance depends on the inputs only instead of the observed targets.

The mean prediction provides a point estimate for the target value of a new test instance. The aim of the experiments presented later in the thesis is to optimize the parameters of Gaussian process regression so that the accuracy of these point estimates is maximized, based on minimizing their squared error. There are two parameters in the expression for the mean: the kernel function, which we assume to be an RBF kernel parameterized by γ, and the Gaussian noise parameter σ. These need to be chosen appropriately for each dataset to maximize the accuracy of the point estimates, and the various optimization techniques investigated in this thesis will be applied to and compared in relation to this problem.

2.2 Decision Trees

In the classification domain of supervised learning, decision tree learning has an important position: it is one of the most widely used and practical methods for categorical target problems. Decision tree learning builds rules for classifying data on the basis of attributes. Table 2.1 provides an example dataset including 14 instances, which is made up of four attributes and the class category. These four attributes are the variables outlook, temperature, humidity and windy, which are related to the weather. The class category has two choices: whether to play, or not. People will make the decision to play or not on the basis of the weather conditions. It is not easy to perform the decision making merely on the basis of the raw table because people have to search every line of the table to find one that matches the current weather situation. Suppose there are thousands, or even millions of records in a table; it is very difficult to find

Table 2.1: The Weather Data

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

the right one in a short time. Moreover, we may not be able to find an exact match. This is where decision tree learning comes to our rescue. A decision tree gives people a graphical form based on a tree structure instead of a raw table. Figure 2.1 below shows a common format for a decision tree for the "weather" dataset, giving a summary of all instances recorded in Table 2.1, which makes it easy for users to find the outcome according to the observed attributes.

According to different splitting rules used for learning, the data sample can generate different decision trees, some of which are too closely fit to the training samples and produce poor predictions, whereas others can classify instances into correct categories and obtain very high accuracy but are too complicated to be understood by everyone. If the user does not mind computing cost and complexity, every attribute of the data could be given the opportunity to be treated as the root node and decision nodes of the decision tree. The resulting decision tree models could be totally different in terms of tree size, depth and outcomes of classification.

In contrast to this exhaustive procedure, all practical decision tree learners have the same form, which starts with a root node, using a top-down search method to split each node recursively.

The nodes of the decision tree, except the root node, include two types: leaf nodes and decision nodes. A leaf node assigns a classification, which contains the class name and the expected value or category of the output; and the decision

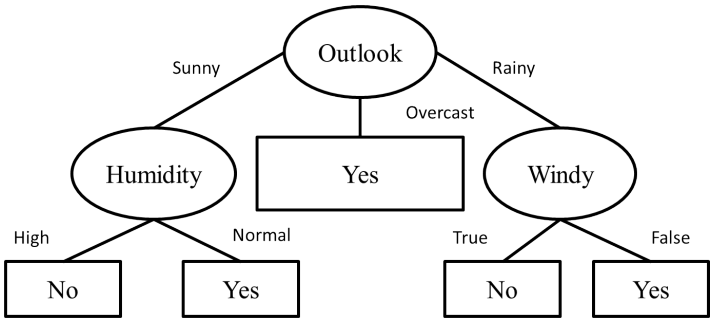


Figure 2.1: A decision tree for the "weather" dataset

node is normally an attribute test with each branch being a possible value of the attribute. The final result of a decision tree learner represents a possible choice according to its learning heuristics.

Decision tree learning is suited to problems having the following characteristics: instances are presented as attribute-value pairs, no matter whether the values are numeric or discrete; the target function has discrete output values, although it is possible to extend targets to real-valued outputs; and the training data possibly contain errors, noise and missing values [38].

Decision tree learning is attractive for inductive learning for three reasons [46]:

- A decision tree is built on account of training instances, and often makes a good generalization for unobserved instances. These instances are the test instances and exhibited with unknown categorical target values.
- The learning methods are generally efficient. The computational complexity is only slightly worse than proportional to the number of observed training instances in practice.
- As a form of expression in the classification process, decision tree learning has great attraction for users, because it is not only readily understandable, but also "renders the classification process self-evident" [46].

Decision tree learning has been applied in a number of commercial practical implementations, such as medical diagnosis [26] [45], credit risk assessment of loan applications [42], and classification of plant diseases [36].

2.2.1 Information Gain

The key concern is how to choose a comparatively better decision tree model from various tree models that are available. In other words, the task is to optimize the structure of the decision tree. The main goal of the decision tree learner is to try to get the purest leaf nodes. Therefore, there are two questions that need to be solved: how to select the most useful attribute to be the decision node to classify the given dataset and how to build trees with as few paths as possible? Applying the so-called information gain as a splitting criterion is one possible approach. Information gain is one of the most well-known splitting criteria to measure the amount of information in a particular attribute that is based on the "purity" of the resulting successor nodes.

In order to understand the information gain, we should first consider the entropy, a measure used to calculate the information gain. Suppose an attribute X has m values, V_1,V_2,...,V_m, and the probability of each value has been given as P($X = V_1$)= p_1 , P($X = V_2$)= p_2,..., P($X = V_m$)= p_m. Then the entropy of X, $H(x)$ [35] is defined as follows:

$$\begin{aligned} H(x) &= -p_1 \log_2 p_1 - p_2 \log_2 p_2 - \cdots - p_m \log_2 p_m \\ &= -\sum_{j=1}^{m} p_j \log_2 p_j \quad (0 < p_j < 1) \end{aligned}$$

For a two-class classification problem with a positive category and negative category in the target classification, the entropy of $H(x)$ can be written as

$$H(x) = -p_+ log_2 p_+ - p_- log_2 p_-$$

Where p_+ is the probability of positive target objectives, and p_- is the probability of negative target objectives. The relation between $H(x)$ and p is shown in Figure 2.2 below.

When p_+ gets closer to 0, which means the proportion of this category is quite small or only a few instances belong to it, then $log_2 p_+$ will be a negative number that is large in absolute terms, and $-p_+ log_2 p_+$ will be a small positive number which is nearly zero; for the remaining part of the equation, because p_+ and p_- sum to 1, p_- gets closer to 1, but $log_2 p_-$ will get close to 0, so $-p_- log_2 p_-$ will also be nearly zero. So the entropy of the entire result works out to be nearly

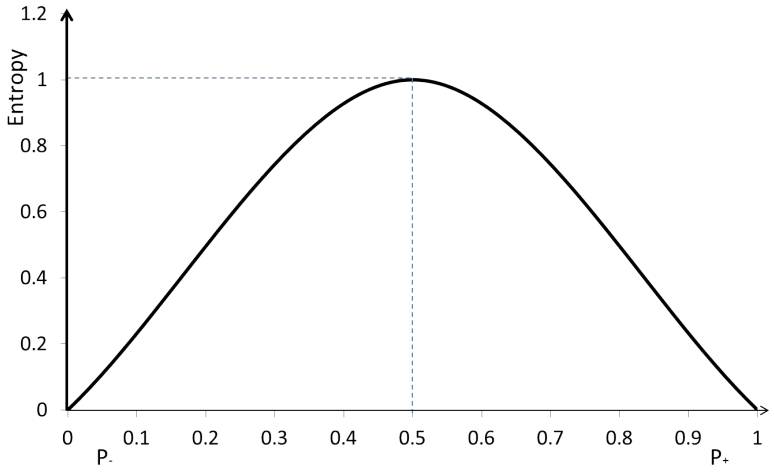


Figure 2.2: The relationship between the entropy and the probability

zero. When both p_+ and p_- are close to 0.5, the set is impure and the entropy is nearly 1. Therefore, a high entropy of X means the distribution is uniform, while low entropy of X means X is from a varied distribution [35].

In the context of decision tree learning, a large change of entropy based on a particular attribute indicates that this attribute should perhaps become the decision node to split the data. We are interested in how much information can be gained due to splitting on this attribute and how to determine the best attribute as the decision node in a tree. Information gain measures the difference between the entropy on the dataset S before knowing the value of the input attribute $H(S)$ and the entropy after splitting the data on the attribute A $H(S, A)$. It can be stated mathematically as follows:

$$IG(S, A) = H(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} H(S_v),$$

where S_v is the subset of S in which the attribute A has value v, $V(A)$ is the set of all values of the attribute A and $|S|$ is the number of instances in S.

2.2.2 ID3

We will use the classic decision tree learning algorithm C4.5 as one of the base learning algorithms in future chapters. Before introducing C4.5, we first consider the ID3 algorithm, which is the predecessor of C4.5. ID3 is an algorithm

to generate a decision tree originally developed by J. Ross Quinlan [21]. ID3 uses a fixed set of examples to build a decision tree and the resulting tree is used to classify future samples. It is used to classify a case by starting at the root of the tree. As with other tree learners, ID3 picks the best attribute for each internal node using a greedy search. If does so recursively until all the instances of a subset fall into the same class - in other words, the final subset is "pure" and becomes a leaf node. Alternatively, if there are no attributes left to consider, tree growth stops. ID3 uses information gain to choose an attribute for each decision node, in order to minimize tree depth. The attribute with the highest information gain is selected because it has the most useful information for classification. The algorithm can be stated in pseudo code as shown in Algorithm 1 [50].

Algorithm 1 ID3(D)[50]

1: $Tree = \emptyset$
2: **if** all the instances belong to one category or D is "pure" **then**
3: a single node with the class value
4: **end if**
5: **for all** attribute $a \in A$ **do**
6: compute information gain
7: **end for**
8: a_{best} = decision node by choosing the highest information gain
9: D_v represents sub datasets from D with the node a_{best}
10: remove a_{best} from the set of instances
11: **for all** D_v **do**
12: $Tree_v = ID3(D_v)$
13: Attach $Tree_v$ to the corresponding branch of Tree
14: **end for**
15: **return** Tree

Assume we have a set of training cases with class values. If all the instances belong to one class, the decision tree process will generate a single node with the corresponding class category or value. Otherwise, the ID3 algorithm will first compute the entropy of the class distribution. The next step is to calculate the

information gain for every attribute. The attribute with the highest information gain will be the root node of the decision tree, and the branches of the root node are based on the values of this particular attribute. Once an attribute is determined as the decision node, this attribute will be removed from the set of instances. The decision tree model is gradually built by repeating this process recursively until all the leaf nodes are pure or there are no more attributes left for splitting.

ID3 enables the user to create easily understandable prediction rules from the whole of the training data. It converges to locally optimal solutions when it searches through the space of decision trees and never performs back tracking. The most important feature of the algorithm is that it can build a small tree in a small amount of training time.

However, if ID3 uses a small sample for training, the tree will probably be over-fitted, and at any given time only one attribute will be selected to make a decision. ID3 may make inaccurate decisions if the dataset contains much noise or the number of training instances is small as it relies greatly on the quality of the training dataset. Also, for continuous attributes, the process is more computationally expensive because of break-point setting.

2.2.3 C4.5 and J4.8

C4.5, which is implemented as J4.8 in the Waikato Environment of Knowledge Analysis (WEKA) software, is an extension of the ID3 algorithm that can be applied to cases that ID3 has difficulty coping with. For data with continuous attributes, C4.5 first sorts the particular attribute according to its corresponding values and finds appropriate thresholds, then evaluates the information gain for every possible position for this split point. After that, it chooses the attribute with the highest information gain. The next enhancement C4.5 makes to the ID3 learning algorithm deals with the problem of missing values. C4.5 splits the corresponding instance with a missing value into pieces, using a numeric weighting method. A very important point is that C4.5 is robust in the presence of noise because it applies a pruning method to avoid over-fitting. The decision tree built by ID3 fits the training examples well but produces poor predictions on test instances on some datasets because it tries to get the purest

leaf node, which makes the tree model too specific and results in the over-fitting problem. It is important to utilize the decision tree model to predict the target values of future instances rather than pursuing optimal performance on the known instances. There are some pruning methods to avoid this phenomenon, such as post-pruning and pre-pruning. Post-pruning adopts pruning measures to discard or replace parts of the decision tree after the tree has been fully grown. Pre-pruning stops growing a branch when a data split is not statistically significant or when the classification becomes unreliable. However, the main disadvantage of this pruning method is that it is difficult to find a stopping threshold. It may terminate division too early and result in early-stopping in the tree growth phase if the threshold is too high; on the other hand, too low a threshold makes the tree pruning negligible, resulting in no significant improvement from the unpruned tree [38].

C4.5 adopts a post-pruning heuristic that is based on computing an error estimate from a statistical confidence interval. The size of this confidence interval is based on a confidence level C, and C4.5 uses the estimated error to determine whether the tree needs to be pruned or not. The default value of C is 0.25, which seems to work reasonably well in many tasks. But if the test error rate of the pruned tree exceeds the estimated error rate, the user can choose lower values of C to yield more pruning.

Another parameter to the algorithm is the minimum leaf size M. The tree will be very large and will have many branches with smaller values of M, and may easily over-fit. But if M is set to be large, then the tree may be too small and there may be poor accuracy in classifying the training data. The pair of parameters (C,M) determines the size of the resulting decision tree. Therefore, it is an important task to find appropriate parameter values for building the tree model using the C4.5 decision tree algorithm, and determining these two parameters is an optimization problem [38].

2.3 Boosting

Boosting is one of the most crucial developments in classification in the machine learning and data mining fields in the last twenty years. Traditional clas-

sification uses a single classifier for prediction. Boosting combines several weak classifiers into a single more complex and more accurate classifier. Early work by Kearns and Valiant [23] indicated that weak learners, each of which performs only slightly better than a random learner, can be integrated to yield accurate performance based on the probably approximately correct (PAC) model. The weak learner is defined as a classifier with a performance that is guaranteed to be better than a coin flip, and a weak classifier will always have a training error smaller than 50% for a dataset with two classes [39]. The weak learner itself will struggle to get high accuracy as the result of the limitations and simplicity of a single classifier system, while multiple classifier systems have been shown to produce favorable performance compared with single-classifier systems [47].

Early boosting theory was rooted in the PAC framework for machine learning, which was introduced by Valiant [48] and Kearns and Vazirani [24]. Kearns [22] first proposed the question whether a set of weak learners can create a "strong" learner to combine the merits of these weak learners and to improve the robustness of the generated learner. Schapire affirmed this was so and developed the first simple boosting approach in 1990, which had significant influence in the development of boosting and machine learning [39]. Three years later, Drucker, Schapire and Simard [7] carried out the first experiment in a practical task (OCR) with the early boosting algorithms. After that, Freund [10] combined many weak classifiers by adopting a "boost by majority" algorithm with the purpose of improving performance. In the context of this boosting theory. One of the best-known boosting methods, AdaBoost (an abbreviation of Adaptive Boosting), was proposed by Freund and Schapire [11]. However, AdaBoost has limitations when coping with noisy data, and Friedman et al. (2000) [12] developed the LogitBoost algorithm that can deal with this problem to a certain extent.

2.3.1 General Algorithm Description

The general idea of boosting is to combine multiple weak classifiers into a "strong" classifier in order to obtain better classification performance. Boosting uses voting in the classification step to combine the output of models. Each individual model (weak classifier) is influenced by the performance of the pre-

model generation

1: Assign equal weight to each training instances
2: For each of t iterations:
Apply learning algorithm to weighted dataset and store resulting model
Compute error e of model on weighted dataset and store error
if e equal to zero, or e is greater or equal to 0.5, terminate model generation
For each instance in dataset:
If instance classified correctly by models, multiply weight of instances by $e/(1-e)$.
3: Normalize weight of all instances.

classification

4: Assign weight of zero to all classes.
5: For each of the t models, add $-log(e/(1-e))$ to weight of class predicted by model
6: Return class with highest weight.

Figure 2.3: Pseudo-code of the General Boosting Algorithm [49]

vious model rather than models being built separately and independently. The method to differentiate the correctly classified and misclassified instances is to assign a set of different weights over the training instances. Boosting encourages new classifiers to become experts to complement the earlier ones which produced misclassification. The way to handle those incorrect predictions is to increase the weight for the corresponding instances, and to decrease the weight for correctly classified instances. This procedure will iterate at each round until some stopping criterion is met. The weight of each instance will be re-weighted after each iteration, which consequently will focus on "hard" instances to be correctly classified by the next weak classifier. The final classifier is a weighted majority vote of the sequence of classifiers. The boosting algorithm (AdaBoost) is summarized in Figure 2.3 [49].

2.3.2 LogitBoost

Before introducing LogitBoost, we will consider the AdaBoost algorithm from above in more detail. AdaBoost is one of the most used boosting algorithms in practical classification prediction. The earliest and simplest AdaBoost

algorithm is the discrete AdaBoost algorithm, which is essentially equal to the AdaBoost.M1 algorithms for two-class classification. Assume we have a dataset with training instances with the format (x_1,y_1),(x_2,y_2),...,(x_n,y_n), where x_i is the feature vector and $y_i = -1$, or 1. The output of the final hypothesis can then be given as the following [40]:

$$F(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

where $\alpha_t = \frac{1}{2} \ln \frac{1-e_t}{e_t}$ and $e_t = Pr_{i \sim D_i}[h_t(x_i) \neq y_i.]$ The AdaBoost procedure trains the individual classifiers on weighted training samples, giving a higher weight to misclassified cases. This process repeats several rounds until e_t is equal to 0 or greater than 0.5. The classifications from each round are then combined linearly.

AdaBoost with trees was regarded as the best "off-the-shelf classifier in the world" in a NIPS workshop [3]. Boosting with trees is commonly used as an ensemble method. Schapire and Singer [41] further developed the AdaBoost algorithm, and extended the discrete predictions to real-valued or "confidence-rated" predictions. More precisely, the weak classifier of each instance outputs a mapping $h_t(x)$: $X \mapsto R$, whose sign is the classification label -1 or 1 and $|h_t(x)|$ is a measure of the "confidence" in the prediction.

AdaBoost can be interpreted as fitting an additive model

$$F(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

to minimize the expectation of an exponential loss function $E(e^{-yF(x)})$ [12]. However, it cannot deal with noisy data very well, which can cause over-fitting problems [14]. The reason is that the exponential loss function changes exponentially as the classification error increases, especially for noisy data samples. But, as mentioned, AdaBoost can be interpreted as an estimation procedure for fitting an additive regression model, which optimizes an exponential criterion. It is possible to replace the exponential function by a binomial log-likelihood criterion, yielding the so-called LogitBoost algorithm [12]. For two-class problems, $y \in \{-1, 1\}$, the expected loss can be converted to $E(-\log(1 + e^{-2yF(x)})$, with $F(x) = \frac{1}{2} log(p(x)/(1 - p(x)))$, where $p(x) = P(y = 1|x)$, which changes the output to be [0,1] and makes it less sensitive to noise.

The pseudo code for LogitBoost for two-classes is shown in Figure 2.4 [12]. LogitBoost is normally applied with regression trees because it requires the base learner to perform numeric prediction. The maximum depth of these trees is an important parameter: using trees that are too complex can lead to over-fitting. To further reduce the likelihood of over-fitting, the predictions of the regression tree are often "shrunk" by multiplying them with a user-specified shrinkage parameter in [0,1]. In real-world problems, we often give a parameter ν to limit the search step $\hat{s}_{m+1}$, which is 0.5 in the LogitBoost algorithm for the two-class case, so the expression can be stated as: $\nu\hat{s}_{m+1}$, $0 < \nu < 1$. This parameter can be seen as a shrinkage parameter. Small shrinkage makes the boosting algorithm learn more slowly and requires a larger number of iterations. For the experiments in this thesis, the regression trees were built using the fast REP tree learner in WEKA. Tree depth and shrinkage are the two parameters being optimized in the experiments. Licamele et al. point out that REP tree can be extremely powerful when it is used in combination with boosting [29].

2.4 Support Vector Machines

In this section, we will consider the support vector machine classifier (SVM), one of the most robust and accurate machine learning methods. To learn an SVM classifier, we will apply the Sequential Minimal Optimization (SMO) [37] algorithm in the experiments presented later, which is a fast algorithm for training SVMs. Before talking about parameter optimization of SVM classifiers, it is necessary to give an overview of the theoretical background.

Let us begin with the simple, linear form of an SVM in a two-class learning task, $y \in \{\pm 1\}$. The aim of the SVM method is then to find a separable hyperplane that can separate positive samples from negative samples by maximum margin (see Figure 2.5). Examples beyond the upper right of the hyperplane are positive examples marked by squares, while negative examples, marked by diamonds, are positioned in the lower left. In this figure, it can be clearly seen that the hyperplane separates the two classes with the margin that maximizes distances to the closest points.

1: Start with weights $w_i = 1/N$, $i = 1,\ldots,N$, $F(x) = \quad 0$ and probability estimates $p(x_i) = 1/2$

2: Repeat for $m = 1,2,\ldots, M$:

(a) Compute the working response and weights

$$z_i = \frac{y_i^* - p(x_i)}{p(x_i)(1 - p(x_i))}$$

$$w_i = p(x_i)(1 - p(x_i))$$

(b) Fit the tree-base learner to the current working response by weighted least squares, using the current weights w_i; the fit function is denoted by $f_{m+1}(x)$.

(c) Update

$$F_{m+1}(x) = F_m(x) + 0.5 f_{m+1}(x)$$

$$p_{m+1}(x) = \frac{e^{F_{m+1}(x)}}{e^{F_{m+1}(x)} + e^{-F_{m+1}(x)}}$$

3: Output the classifier

$$sign[F(x) = sign[\sum_{m=1}^{M} f_m(x)]$$

Figure 2.4: Pseudo-code of the LogitBoost Algorithm for two-class problems [12]

The formula for the linear SVM output can be written as:

$$u(x) = w^T x + b \tag{2.9}$$

where x is the input vector, and w and b are the weight vector and bias of the optimal hyperplane respectively. The separating hyperplane can be regarded as $u = 0$ and the nearest points are on the planes $u = \pm 1$. Hence, the distance

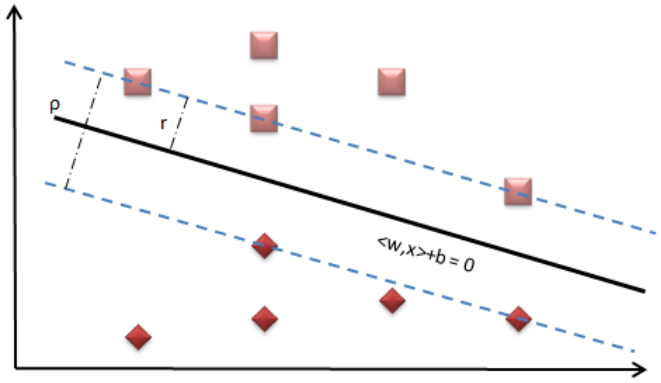


Figure 2.5: A linear support vector machine

from any point x to the separating hyperplane is

$$r = \frac{u(x)}{\| w \|} = \begin{cases} \frac{1}{\|w\|} & if \quad u(x) = +1 \\ -\frac{1}{\|w\|} & if \quad u(x) = -1 \end{cases} \tag{2.10}$$

The margin of separation ρ can be written as follows:

$$\rho = 2r = \frac{2}{\| w \|}$$

Consequently, finding the maximizing margin can be represented via the following optimization expression:

$$\begin{aligned} & min_{w,b} \frac{1}{2} \| w \|^2 \\ subject \quad to \quad & y_i(w^T x_i + b) \geq 1, i = 1, 2, \ldots, n \end{aligned} \tag{2.11}$$

This optimization problem can be solved by the method of Lagrangian multipliers, which converts it into a quadratic programming optimization problem. At the same time, we can derive the relationship between each Lagrange multiplier and each training data point, which is

$$w = \sum_{i=1}^{T} y_i \alpha_i x_i,$$

hence, the classifier function for a new data point x can be written as follows:

$$f(x) = \sum_{i=1}^{N} y_i \alpha_i < x_i, x > +b \tag{2.12}$$

The interesting point of this formula is that the prediction of the new point x is dependent on the inner product of x and training points x_i. It is computationally

complex to calculate every inner product for every training point, but in fact, the α on all the non-support vectors is zero. The non-support vectors are those data points that do not meet the condition $y(w^T x+b) = 1$. Therefore, the inner products involving the new data point are based only on the support vectors rather than on all training data points. The reason is that the classification completely depends on the hyperplane and the irrelevant points have no effect on the classification.

This is the traditional and classic way of training SVMs, but the basic form of the optimization algorithm takes longer to compute than is practical, especially for large training datasets; also, the corresponding algorithms are complex and difficult for an average user to implement. Thus, John [37] proposes the so-called SMO algorithm for training SVMs, which converts this large quadratic programming (QP) problem into a series of smallest possible QP problems and uses an analytic QP step instead of an inner loop.

Unfortunately, not all data points are linearly separable by a straight hyperplane in real-world problems, and it is impossible to solve cases with a few noisy data points merely using the basic maximum margin algorithms. If some of the positive samples and part of the negative samples are mixed together, that is, a non-zero training error exists even for the maximum margin hyperplane, the question becomes how to separate them in order to reduce the error as much as possible. Figure 2.6 shows an example. The original hyperplane cannot divide these two sets of examples very well because noisy data exists. In order to distinguish the two sets, the hyperplane has to move to the black dotted line, which leads to the margin of the new hyperplane becoming smaller. The situation will become worse if the outlier s move further left, and these two sets of examples will mix together. Then, it is impossible to find a straight line to separate them. Outlier data points can be ignored in some cases where they have no effect on the classifier, while sometimes these data points are essential because they are support vectors.

The "soft margin" introduced by Cortes and Vapnik [5] is a solution to extend SVMs in order to allow a few noisy points to be misclassified. To achieve this, there is a parameter C that trades off the margin with the number of inseparable points, based on another set of parameters $1 > \xi_i > 0$ that allows

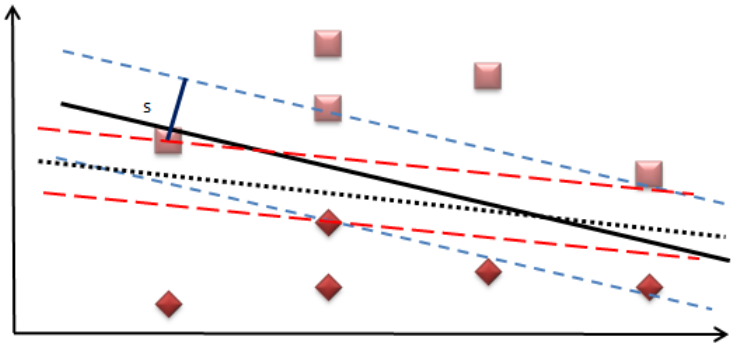


Figure 2.6: Support vector classification with outlier

the margin constraints to be slackened. The expression of the optimization problem can then be stated as:

$$\begin{aligned} & min_{w,b,\xi} \tfrac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{l} \xi_i \\ & subject \quad to \quad y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1, 2, \ldots, n \end{aligned} \tag{2.13}$$

The value of C varies with the dataset used and the optimal performance is often assessed by evaluation, e.g. cross-validation in many cases. The change of the parameter C will directly affect the value for $\parallel w \parallel^2$ and the level of the other parameter, ξ. In other words, with the fluctuation of C through a series of values, the minimum result moves to the corresponding outcome. It can be seen that a larger value of C will lead to smaller ξ in this optimization problem.

By adapting the same method mentioned above, the soft margin problem can also be solved using Lagrange multipliers α_i [37].

$$\begin{aligned} min\Psi(\alpha) = min \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j - \sum_{i=1}^{n} \alpha_i \\ subject \quad to \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C, i = 1, 2, \ldots, n \end{aligned} \tag{2.14}$$

This "soft margin" algorithm solves some real-world problems to a certain degree, especially when only a few noisy data points exist and under the condition that users do not require high performance. However, when the problem is nonlinear or exhibits heavily linearly inseparable structure, it is not ideal to insist on using the preceding approach as it will produce many classification errors [50].

Thus, Cortes and Vapnik [5] considered a proposition that maps the given data into a feature space with higher dimension and then finds the optimal

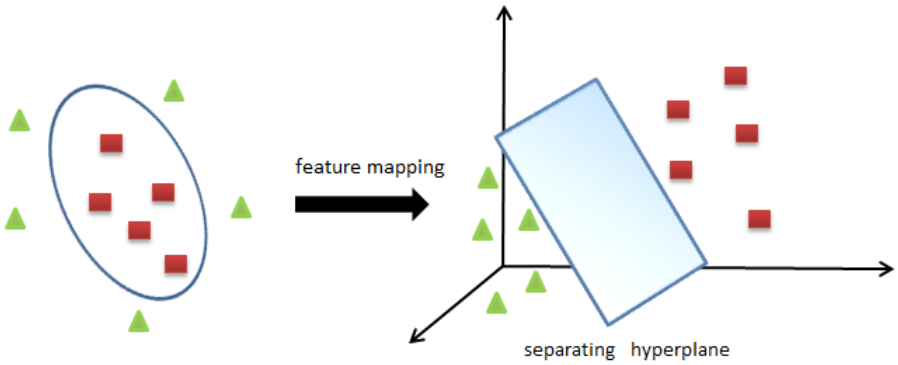


Figure 2.7: The mapping from the original space to the feature space

hyperplane in this feature space. Figure 2.7 illustrates the mapping from the original low dimensional space to higher dimensional space. The left part shows that the given data points are not linearly separable originally but become linearly separable after mapping into the higher dimensional feature space.

This process is a transformation between the original space and the higher dimensional space in order to make the problem linearly separable. The key to the transformation is a "kernel function", which has been discussed earlier in this chapter in the context of Gaussian process regression. The optimization problem for the soft-margin SVM combined with a kernel function can be expressed as [6]:

$$\begin{aligned} min\Psi(\alpha) = min\tfrac{1}{2}\textstyle\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K(x_i, x_j) - \sum_{i=1}^{n}\alpha_i \\ subject \quad to \quad \textstyle\sum_{i=1}^{n}\alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C, i = 1, 2, \ldots, n \end{aligned} \tag{2.15}$$

Furthermore, the optimal classifier can be written in the following form:

$$f(x) = \sum_{i=1}^{n} \alpha_i^* y_i K(x_i, x_j) + b^* \tag{2.16}$$

Many kernel functions can be used for the above settings, but only a few functions work well in a wide variety of applications. In this thesis, we use only the RBF kernel function, which is a recommended function for the SVM

classifier. Hence, Equation 2.14 can be rewritten as:

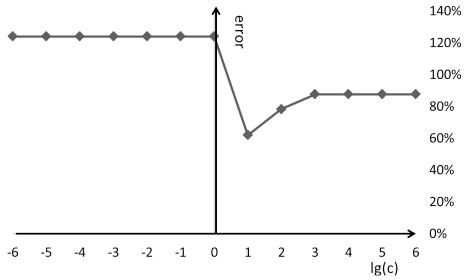
$$\begin{gathered} min\Psi(\alpha) = min\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j exp(-\gamma\|x_i - x_j\|^2) - \sum_{i=1}^{n}\alpha_i \\ subject \quad to \quad \sum_{i=1}^{n}\alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C, i = 1, 2, \ldots, n \end{gathered} \tag{2.17}$$

Given this expression, and an algorithm such as SMO for finding a solution in an efficient way, the problem we address in this thesis is then to find the optimal pair of parameter settings (C, γ), and thus to get the best support vector classifier.

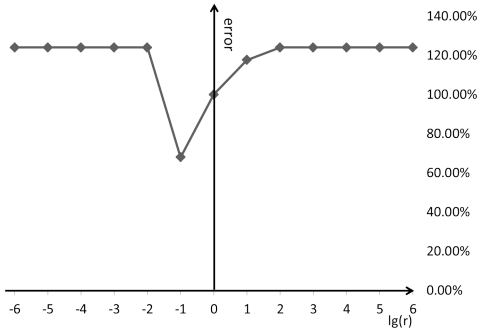
Figure 2.8 gives us trends of root relative squared error with the values of these two parameters increasing on the "labour" dataset from the UCI repository. These sub-figures exhibit the influence of one parameter on the error when the other parameter value is fixed. Figure 2.8(a) shows the relationship between different values of the parameter C and their corresponding error performance. It seems that there are three stages for this trend: initially, it is at a stage before arriving at zero; next, it has a deep decrease until reaching one; and then it increases slowly and remains stable after that. Figure 2.8(b) has a similar trend as one noted above, but with two main differences. The lowest point is different from that shown in Figure 2.8(a): the error is lowest when the parameter γ is 0.1. Another difference is that it reaches a higher level of error at the later stage. Therefore, finding a good parameter setting is a key step for building the classification model, but unfortunately the "best" pairs of parameters vary for different problems. Every pair of parameters corresponds to a potentially different classification performance. The best performance will correspond to the best parameter combination.

2.5 Cross-Validation

The standard method is to use cross-validation when determining tuning parameters. Cross-validation is a powerful tool to evaluate the effectiveness of the chosen parameters. John investigated the use of cross-validation to select parameters for the C4.5 decision tree learning algorithm, which showed that the accuracy of the induced trees on independent test sets is generally higher after applying cross-validation than the accuracy when using the default parameter



(a) The relationship between lg(c) and error



(b) The relationship between lg(γ) and error

Figure 2.8: Relationships of parameters and errors

value [19]. Cross-validation is used for support vector machines as well. The paper [34] considers leave-one-out cross-validation to estimate the measure of goodness-of-fit in SVMs, which provides estimates of the bias of the excess error in a prediction rule constructed with training samples.

Cross-validation is a statistical method for estimating the predictive performance of models by dividing data into a certain amount of data for training a model and the rest for testing. We always use this approach to measure the prediction performance of the model learned from training data on future data with unknown values. The original data can be divided into a training set and a validation set. The model is learned from the training set repeatedly and we test its performance on the remaining validation set. Normally, the subset used for training is bigger than that for testing. It is better to use more than half of the data for training. For example, one could use two-thirds of data randomly selected for training and the remaining one-third for testing [49].

In cross-validation, the partition of the data can be specified by the user.

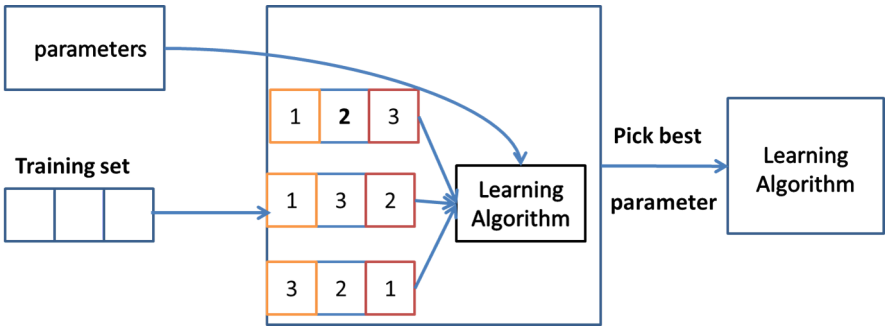


Figure 2.9: Cross-validation methodology

More partitions of data mean more precise performance estimates and more time spent on running the procedure. The basic form of cross-validation is v-fold cross-validation, that is, the whole dataset is partitioned into v parts of equal size randomly [25]. First of all, a group of samples is randomly chosen as a validation set, while other groups of datasets are used to build the model. Next, the samples from the validation set are used to test the model's predictive performance. Then, we use the same method for the other groups in turn by training the model on the remaining $v - 1$ groups. The final performance will be the average of all the performance scores of all the groups, since every group has its own performance score. Therefore, cross-validation requires that the machine learning model be trained multiple times in order to obtain an average performance score for a parameter setting [25].

We give a simple illustration to describe how cross-validation works to pick the comparatively "best" parameter in Figure 2.9. In this figure, the parameter or the combination of parameters to be evaluated is determined in advance. The training set is divided into three blocks, and we repeatedly train the learning algorithm using two of these three blocks as the "model construction set" and the remaining one as the "validation set" to estimate the performance. We can run this as many times as specified by the user and get the mean/median of the performance criteria to get the best parameter. To simplify the process, the figure is based on three runs only, which is also called "3-fold cross-validation".

Although the number of folds in cross-validation is specified by users in the experiment set-up, we generally use 10-fold cross-validation to test the performance of certain algorithms. This is well accepted as the standard way of

predicting performance given a fixed sample of data. Thus the learning scheme is executed 10 times on 10 subsets and an overall performance can be obtained by averaging the 10 estimates. To obtain better estimates in some experiments, it is a common procedure to repeat 10-fold cross-validation 10 times, which gives 100 results in total, and then average them. As a result, cross-validation can be used to help prevent a model being over fitted. In short, this technique is a standard tool for helping users develop and fine-tune data mining models.

2.6 Summary

In this chapter, we first considered the mathematical background of Gaussian processes. Gaussian processes have some attractive characteristics, which provide the global distribution for an unknown data point rather than just the prediction of a single value. Then, we introduced the algorithms as well as the parameters that will be optimized in the following chapters, including common and well-known algorithms, such as decision tree learning, support vector machines and boosting. The final section provided a description of the cross-validation method, a procedure designed to evaluate the parameters of a learning scheme in order to determine optimal values for those parameters.

Chapter 3

Gaussian Process Optimization

A promising way to perform parameter optimization in machine learning, is to use data for which performance scores have been collected to build a predictive model, and then to select points from the search space. The search space can be 2-dimensional or multi-dimensional depending on the number of parameters to be optimized. This method is potentially effective when the data size is large and the cost of applying the learning algorithm to be optimized is also large. In contrast, in cases that need limited time to run the learning algorithm, perhaps on a small dataset, it is efficient to use relatively inexpensive resources to apply a more brute-force approach to finding appropriate parameter settings.

In this chapter, we will apply a suitable predictive model to search the parameter space for likely candidate points, and evaluate them with the criterion specified by the user. The model used for this task has a full predictive distribution rather than a single prediction at each candidate point. This methodology enables us to take predictive uncertainty into account when exploring the search space instead of simply fitting the existing training data and obtaining the point with the largest (or smallest) prediction.

The Gaussian process model that we apply for that purpose has distinct advantages from this standpoint because it enables us to obtain the predictive distribution easily and learn general properties of the learning algorithm's

behaviour from previous samples by using an appropriate kernel function [9]. Based on this model, the chosen point of the search space is set where the expected improvement in the learning algorithm's performance is highest. We will also introduce a stopping criterion to stop the search procedure when further expected improvement is likely to be small. The best point is then chosen from all the points evaluated so far. We will use this pair of parameters to build our final optimized model and thus use it to predict the unknown target value of new instances with respect to their known attributes.

However, before discussing the Gaussian-process-based optimization method in more detail, let us first briefly discuss two simple optimization methods that will be used as baselines for the experiments in this thesis: Grid Search (as implemented in the WEKA software), and Random Search.

3.1 Search Methods

The search methods to be used in this thesis includes Grid Search and Random Search. These search methods are both based on the same general idea, which is shown in Figure 3.1.

Both methods start with a pair of initial parameter values, and acquire the corresponding performance based on a certain learning algorithm. The following step is to generate a new data point (new parameter values) based on the search method and to evaluate its performance. The new performance will be compared with the old one. If the performance is improved, then we update the parameter value in the parameter range. This process continues repeatedly until there are no more improvements. The best pair of parameters is decided in the final step. We will explain the different procedures of Grid Search and Random Search in the following two sections.

3.1.1 Grid Search

Grid Search is a simple search methodology to identify appropriate parameter values for a machine learning method, where the points are located on a grid in the parameter space. Any point in the grid has the opportunity to be chosen as the best parameter combination. This method has the obvious ad-

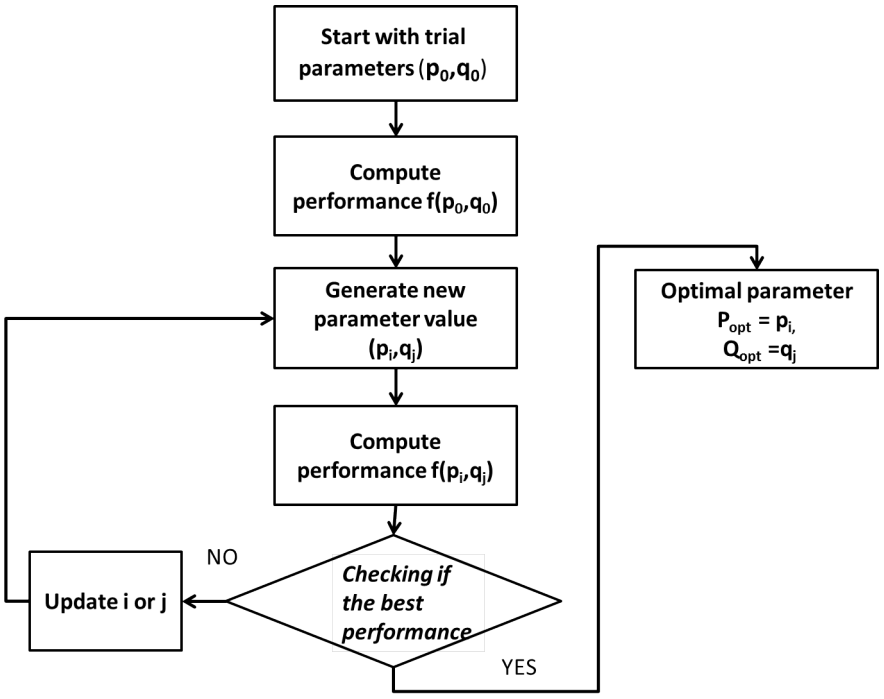


Figure 3.1: Flowchart of general search algorithm for two parameters

vantage that it does a complete search of every point in the parameter grid that is specified by the user, and one can usually find good parameters after this brute-force search method. In detail, Grid Search involves setting up a grid of parameter combinations in the space and evaluating the values of the objective function at each grid point. The point corresponding to the best value of the objective function is considered to be the optimum solution.

There are two drawbacks to this method [28]: one is that the number of grid points increases exponentially with the number of decision variables, which makes it computationally very costly; the other is that it is generally inefficient. Suppose the given dataset has millions of instances or thousands of attributes per instance and the search domain is large, with many grid points. Grid Search will waste a great deal of time on unnecessary points.

The Grid Search methodology has been implemented in the WEKA software. In this interface, the user can specify the parameter values, the search increments (precision), the lowest value and the highest value for each parame-

ter, and the search direction: whether it will search by rows or columns. Once the experimental set up has been completed, the grid of data points is searched according to the specified direction. The increment determines the distance between neighbours: the smaller the increment, the finer the grid. This method explores the space constrained by upper and lower bounds, where the upper bound is the highest value of one parameter, and the lower bound is the lowest value of the other parameter. It starts from the lowest value of the pair of parameters, moving in an appropriate direction during each increment until the last point in the grid. During this process, every point will be validated by 2-fold cross-validation and evaluated by the specified criterion, such as accuracy or error. After that, the initial best point (parameter value) will be found by comparing the performance scores obtained.

However, the above process is a coarse search process. Once an initial best point has been found, a new nine-point grid is centred on this point and evaluated using 10-fold cross-validation. This latter step is repeated when a new best point is discovered in the nine-point grid. The optimization is completed when the grid becomes centred on a point with the minimum performance or a point with the maximum performance, which depends on the evaluation of the objective function.

3.1.2 Random Search

An alternative to Grid Search is Random Search. The Random Search method can be regarded as a stochastic search process. This method generates solutions randomly for the model being optimized and finds the best solution by comparing the corresponding objective function values. In the experiments of this thesis, we specify a certain number of data points generated by the random number generator. The data points here are actually pairs of parameters which we have set in advance based on a grid - the same one as in Grid Search. For any parameter, the user can set its minimum value and maximum value empirically. For example, assume the parameter x has m possible values and the parameter y has n possible values, thus pairs of these two parameters will have $m * n$ combinations. The random number generator will choose a pair of parameters among them randomly. Random Search has an obvious advantage in terms of

the cost of computation. In Chapter 4 we will see that Random Search has better performance than the other methods considered in this thesis on some datasets. However, the number of points to be evaluated needs to be specified by the user.

3.2 Expected Improvement

Rather than using Grid Search or its randomized variant, we want to consider using a Gaussian process model to search more intelligently. The key component here is the so-called expected improvement, which we will discuss first. Because the predictive distribution at any test point can be calculated by the model built with existing points, we can use it to estimate how much improvement we can expect at any candidate point. Here, we will adopt the Expected Improvement (EI) to measure the improvement which is defined by Johns [20] in the EGO optimization algorithm. The basic idea underlying EI is to estimate the improvement of possible candidate points over the current best value and to sample at the particular point next where the expected improvement is maximum. Based on this approach, the algorithm can sample points in the parameter space that the user has specified, and use the information it has learned to decide where to sample next. Fortunately, the expected improvement is easy to use and calculate when using a Gaussian process model.

We now discuss how EI can be calculated, closely following the exposition in [9]. The predicted improvement $I(x^*)$ at some test point x^* in the parameter space is defined as

$$I(x^*) = max((p(x^*) - f_{max}), 0)$$

in a maximization problem, where f_{max} is the current best of the existing values and $p(x^*)$ is the prediction at x^* based on the model, which in the case of a Gaussian process model is Gaussian distributed as $p(x^*) \sim \mathcal{N}(\hat{\mu}(x^*), \hat{\sigma}(x^*)^2)$. The function $p(x^*)$ is a random variable that models the uncertainty at x^*, and is treated as a normal density function with mean $\hat{\mu}(x^*)$ and standard deviation

$\hat{\sigma}(x^*)$. The expected improvement can be obtained as follows:

$$\begin{aligned} E[I(x^*)] &= E[max(0, I(x^*)] \\ &= \int_{I=0}^{I=\infty} Ip(I)dI \\ &= (\hat{\mu}(x^*) - f_{max})\Phi(\frac{\hat{\mu}(x^*) - f_{max}}{\hat{\sigma}(x^*)}) \\ &+ \hat{\sigma}(x^*)\phi(\frac{\hat{\mu}(x^*) - f_{max}}{\hat{\sigma}(x^*)}) \end{aligned} \tag{3.1}$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution and $\phi(\cdot)$ is the standard normal density function. The quantities $\hat{\sigma}(x^*)$ and $\hat{\mu}(x^*)$ can be obtained from the Gaussian process model, as explained in the previous chapter.

For a minimization problem, the predictive improvement $I(x^*)$ at some point x^* is defined as

$$I(x^*) = max(0, (f_{min} - p(x^*)))$$

where f_{min} is the current minimum value. Therefore the equation above will change as follows [20]:

$$\begin{aligned} E[I(x^*)] &= E[max(0, I(x^*)] \\ &= \int_{I=0}^{I=\infty} Ip(I)dI \\ &= (f_{min} - \mu(\hat{x^*}))\Phi(\frac{f_{min} - \hat{\mu}(x^*)}{\hat{\sigma}(x^*)}) \\ &+ \hat{\sigma}(x^*)\phi(\frac{f_{min} - \hat{\mu}(x^*)}{\hat{\sigma}(x^*)}) \end{aligned} \tag{3.2}$$

3.3 Gaussian Process Optimization

Gaussian Process Optimization (GPO), as considered in this thesis, is an optimization process maximizing the expected improvement based on Gaussian process predictions associated with a grid of candidate points. It is considered as an alternative to a simple Grid Search to find the parameters of the learning algorithm to be optimized. We assume that there are two parameters that are to be optimized simultaneously. Our goal is to use as few points as possible to find the best pair of parameter values in a search space, where the crucial point is to update the Gaussian process model with a new performance observation

in each iteration. In other words, the training model changes in each iteration as a new data point arrives.

The Gaussian Process Optimization method can be described in detail as follows. First, our GPO method needs to collect some initial data points for the initial Gaussian process model to be trained on. We do this by running cross-validation for the four "corners" of the parameter space, whose positions are specified by the user, using the base learning algorithm and the dataset concerned. Second, the method builds a Gaussian process model from this initial training data. The collected training data consists of two attributes representing the parameter values and the target which is the performance score obtained from cross-validation. Then, the "best" point in the parameter space can be calculated in terms of maximizing expected improvement using the GP model.

Once the "best" point has been found after each iteration, the algorithm will run a cross-validation using this pair of parameter values in order to obtain the corresponding performance. This so-called "best" point becomes a new training point for updating the Gaussian process model with its performance score as y value. With an increasing number of data points used for training, this process builds a model that more and more closely reflects the real function.

The process is applied iteratively: we find the next "best" point in the parameter space using the same method as above until the stopping criterion has been reached and compare the performance among those evaluated points to look for the actual best one. In the implementation used in this thesis, 2-fold cross-validation is used to obtain the performance estimates in this search process.

The "best" point found through the above steps is likely to be close to the real one. However, finally, we will perform 10-fold cross-validation at this point and its eight neighbours, i.e. at nine points, to try to find out whether there are even better points in the neighbourhood. This is done because 2-fold cross-validation is fast but is more likely to miss the "true" best point than is 10-fold cross-validation. This is the same process as in simple Grid Search and Random Search, discussed in the previous sections. This search will stop when no better points are found or the best one is on the border of the parameter space. In the final stage, the algorithm will build the final model for the base learning

algorithm on the user-specified data using the best parameter combination found using the above steps.

The pseudo code for the GPO algorithm as applied in this thesis is shown in Algorithm 2.

3.4 The Effect of the GP Parameters

For any particular dataset, users need to choose the base learning algorithm as the classifier to use, and specify parameters which they want to optimize. These parameters can be parameters of the base learner combined with parameters of base learner components. For example, when optimizing parameters of a Gaussian process algorithm as the base learning algorithm, parameters can include the noise level parameter and the kernel parameter. However, the GPO method itself uses a Gaussian process model with noise parameter δ_{GPO} and the parameter of the kernel function, such as the RBF kernel parameter γ_{GPO}, both values of which can be assigned by users.

To present the behaviour of GPO, we will search for pairs of parameters (δ, γ) of a Gaussian process algorithm as the base learning algorithm, where δ_{base} is the noise level of the classifier and γ_{base} defines the width of the RBF kernel in the classifier (although σ_{base} is actually the width of that kernel parameter). The search range for each parameter is specified on an exponential scale with exponents from -3 to 3 with step size 1. Hence, the parameter values will be $\delta_{base} \in \{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$ and $\gamma_{base} \in \{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$. In this search space, there are 45 points in total, excluding the four corners of the space, which are $(\delta_{base}^{min}, \gamma_{base}^{min})$, $(\delta_{base}^{min}, \gamma_{base}^{max})$, $(\delta_{base}^{max}, \gamma_{base}^{min})$, $(\delta_{base}^{max}, \gamma_{base}^{max})$. In this case, 10^{-3} is the minimum value and 10^{3} is the maximum of both ranges. The four points with their performance scores are treated as initial training data points, and are used for building the initial GP model to predict the likely best point in the grid by finding the highest expected improvement. The number of remaining grid locations decreases in each iteration, in that the chosen point in each iteration will not take part in the calculation for the next looping. Each new point has a predicted value obtained by the model built from the existing points.

We now consider the first 10 iterations of the search process as examples to

Algorithm 2 The GPO algorithm (excluding iterated Grid Search around "best" point based on 10-fold cross-validation)

Input: X_{min} , X_{max} , Y_{min} , Y_{max}

Output: predicted value (P), standard deviation (SD), expected improvement (EI), actual value (A)

1: $d_1 \leftarrow (X_{min}, Y_{min})$, $d_2 \leftarrow (X_{min}, Y_{max})$, $d_3 \leftarrow (X_{max}, Y_{min})$, $d_4 \leftarrow (X_{max}, Y_{max})$ as the four corners of the grid
2: **for** $i = 1 \rightarrow 4$ **do**
3: $p_i \leftarrow getPerformance(d_i, 2FoldsCrossvalidation)$
4: $i \leftarrow i + 1$
5: **end for**
6: Training Data: $D \Leftarrow (d_i, p_i)$
7: **repeat**
8: Create a new Gaussian process object with specified γ and δ parameters
9: Build a GP model on D
10: $N \leftarrow 0$
11: **while** $N \neq$ the number of points in grid **do**
12: **if** Maximization problem **then**
13: $EI \Leftarrow ExpectedImprovementMax(GP, d_N)$
14: $N \leftarrow N + 1$
15: **else** {Minimization problem}
16: $EI \Leftarrow ExpectedImprovementMin(GP, d_N)$
17: $N \leftarrow N + 1$
18: **end if**
19: **end while**
20: $EI_{best} \Leftarrow max(EI_N)$
21: d_{new} is the point of EI_{best}
22: $A_{new} \leftarrow getPerformance(d_{new}, 2FoldsCrossvalidation)$
23: **until** stopping criterion satisfied
24: $d_{best} \Leftarrow max(d_N)$
25: $A_{best} \leftarrow getPerformance(d_{best}, 10FoldsCrossvalidation)$

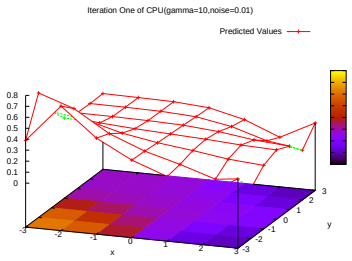
illustrate the behaviour of GPO based on four measures. The following figures show the plots of the predicted value, standard deviation, expected improvement and actual value on the "CPU" dataset. In addition, we set the RBF kernel parameter of the GP model γ_{GPO} as 10 and the Gaussian process noise δ_{GPO} as 0.01. Note that we have found empirically that it is crucial to choose an appropriate pair of parameter values (δ_{GPO}, γ_{GPO}) for the GP model in the optimization process.

Figure 3.2 shows the predicted values for the first 10 iterations. The grid diagram at the top of each plot shows how the predicted values change, and the colour on the bottom plane surface indicates the magnitude of the predicted values. The brighter the colour, the larger the value. We can easily identify movements when putting the 10 iterations together.

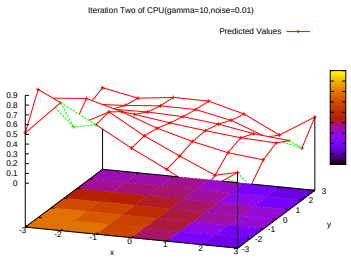
The plot of Iteration i is relatively flat because no new points are added except the four points on the Z axis that show the values of the four "corners" in parameter space. The plots start to fluctuate as new points arrive and the GP model for the predictions grows more precise as the iterations increase. Greater fluctuation always occurs when the upcoming point is located far away from the last existing point. If the plots of two iterations look similar, this implies that the points fall into the same sub area of the grid.

Standard deviation measures the uncertainty at a certain point x. A local point with large standard deviation implies substantial uncertainty at that point, while lower standard deviation implies less uncertainty. Figure 3.3 shows the standard deviation in the first 10 iterations. The central area has higher standard deviation values in the first plot in that there is not enough information and large changes may follow. In contrast, the corner areas have lower standard deviation values because known values are located there. The standard deviation at some point x turns very small and close to 0 once this point has already been chosen, and it gradually drops across the whole grid with new added points until the tendency of the plots becomes flatter in the end.

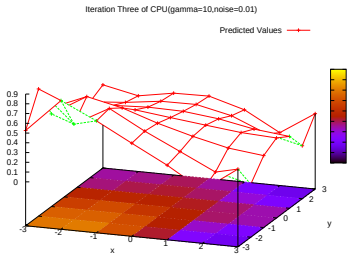
One key method in the GP model for the GPO approach is to calculate expected improvement. The expected improvement is a crucial step to decide which point will be sampled and added into the training data to build the updated model. This approach will search the whole space to look for the point



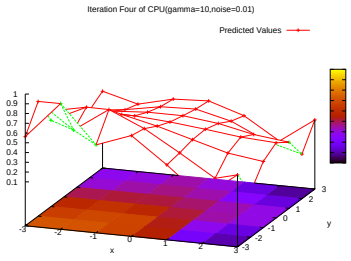
(a) Iteration i



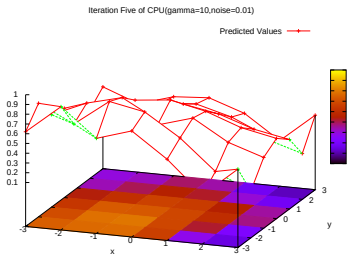
(b) Iteration ii



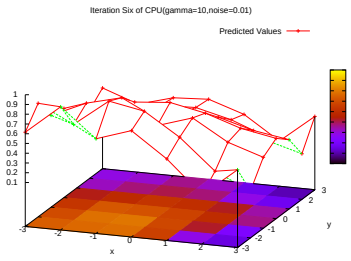
(c) Iteration iii



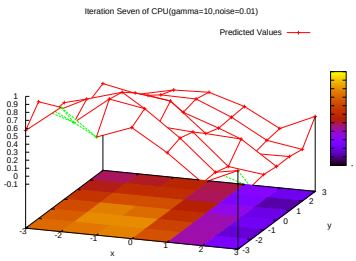
(d) Iteration iv



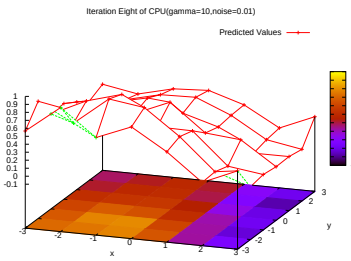
(e) Iteration v



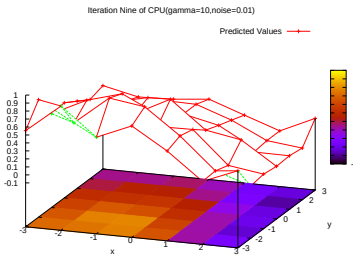
(f) Iteration vi



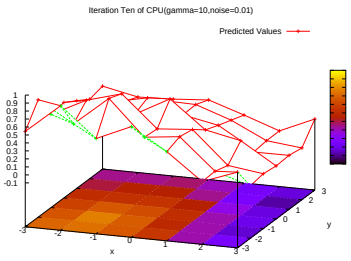
(g) Iteration vii



(h) Iteration viii

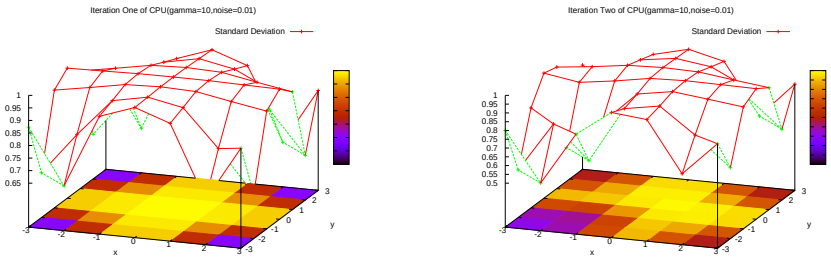


(i) Iteration ix



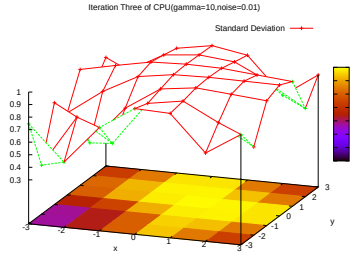
(j) Iteration x

Figure 3.2: Predicted Values with $\gamma_{GPO} = 10$ and $\delta_{GPO} = 0.01$ in 10 iterations

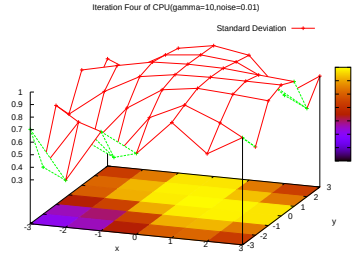


(a) Iteration i

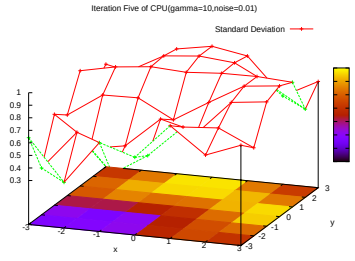
(b) Iteration ii



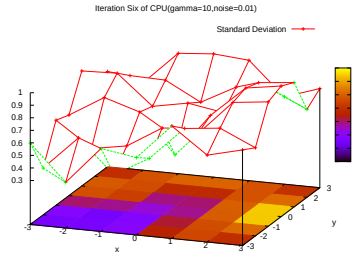
(c) Iteration iii



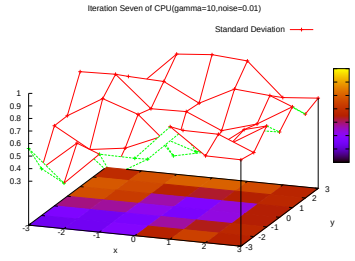
(d) Iteration iv



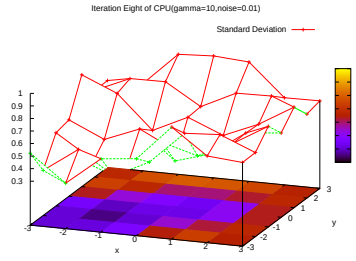
(e) Iteration v



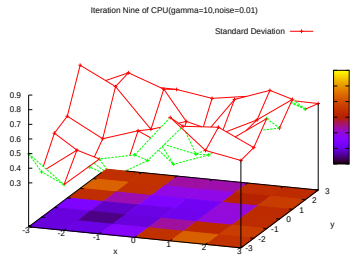
(f) Iteration vi



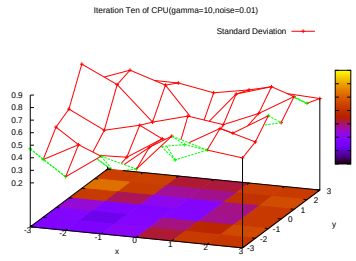
(g) Iteration vii



(h) Iteration viii

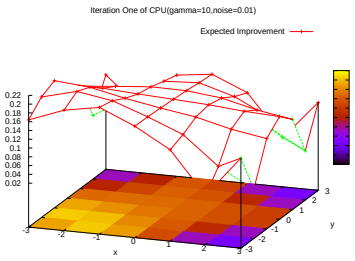


(i) Iteration ix

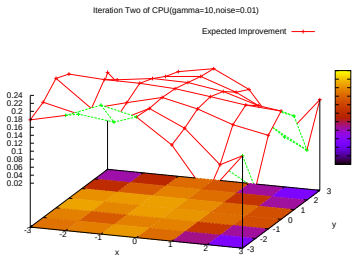


(j) Iteration x

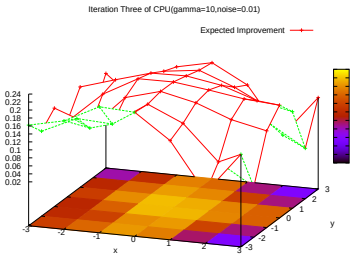
Figure 3.3: Standard Deviation with $\gamma_{GPO} = 10$ and $\delta_{GPO} = 0.01$ in 10 iterations



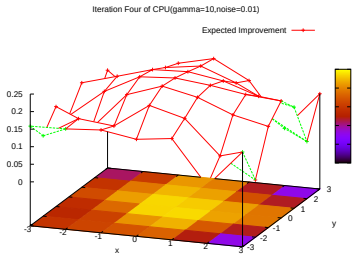
(a) Iteration i



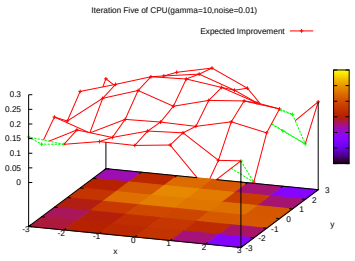
(b) Iteration ii



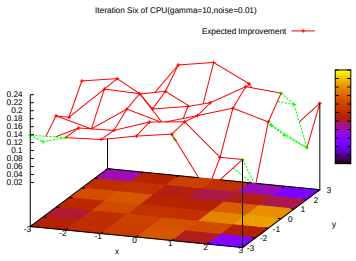
(c) Iteration iii



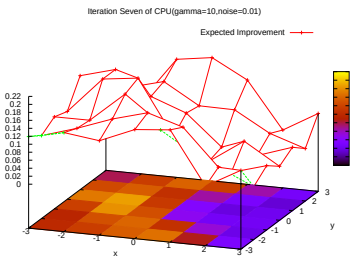
(d) Iteration iv



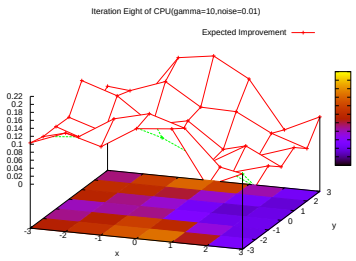
(e) Iteration v



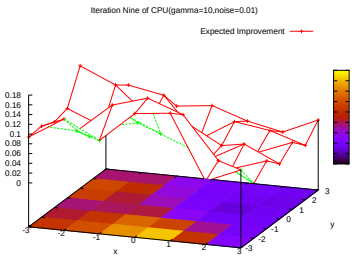
(f) Iteration vi



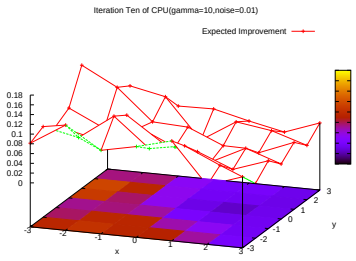
(g) Iteration vii



(h) Iteration viii



(i) Iteration ix



(j) Iteration x

Figure 3.4: Expected Improvement with $\gamma_{GPO} = 10$ and $\delta_{GPO} = 0.01$ in 10 iterations

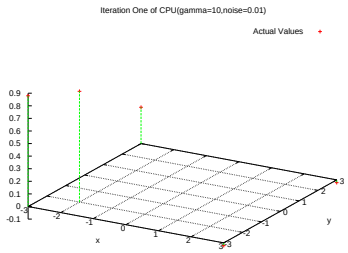
with maximum expected improvement. The point that meets this requirement will be the next candidate sample. Then, this sample coupled with its performance will be added into the training set to update the model for sampling the next point. The expected improvement of the first 10 iterations is shown in Figure 3.4. In this figure, we have 10 points in total since a point is chosen based on the maximum expected improvement in each iteration.

Once an appropriate point has been decided, its actual value is evaluated using cross-validation. It is easy to sort these actual performance scores in order to obtain the best performance, whose corresponding parameter values will be used to build the final model. Based on the expected improvement shown in Figure 3.4, we can easily identify the next most promising point in the specified search space and obtain actual performance values through cross-validation. Figure 3.5 shows the positions and values of actual points in the first 10 iterations. The red mark indicates the real position of the actual sample in the space. The dots mapped on the bottom grid demonstrate the pairs of parameter values clearly. The best performance can be found by comparing the actual values of these points.

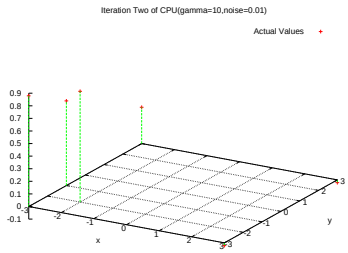
Apart from the above values for the GPO parameters γ_{GPO} and δ_{GPO}, we also did the same experiment based on different pairs of parameters. γ values in the range from 0.01 to 100 were considered and the level of noise δ was set to be 0.1 and 1. Some further example plots are shown in Appendix A and B. Comparing all the experimental results obtained, we always observed slightly better performance and quicker convergence to the optimal point when γ and δ were set to 10 and 0.01 respectively, as in the plots discussed above. This was also the case for other datasets that were considered.

3.5 Stopping Criterion

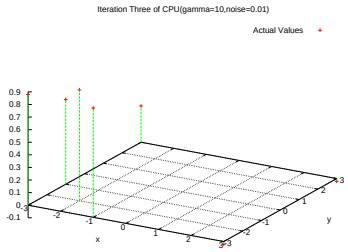
Comparing GPO with Grid Search, the main difference is that Grid Search has to do a brute-force search exhaustively while GPO searches only the area where it is likely to obtain better performance. In Grid Search, nothing to necessitate making any prediction for the next sample as the search direction and the search step is specified by the user beforehand. In contrast, GPO



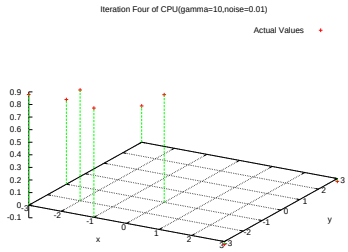
(a) Iteration i



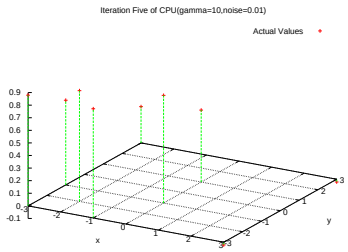
(b) Iteration ii



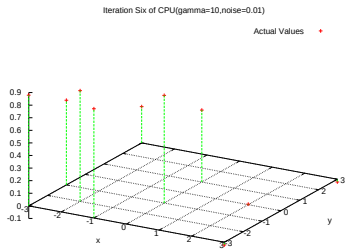
(c) Iteration iii



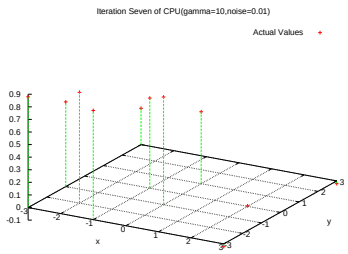
(d) Iteration iv



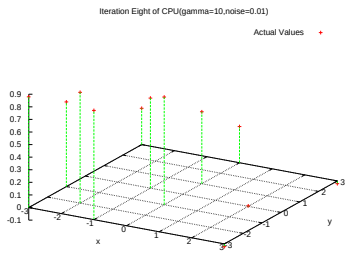
(e) Iteration v



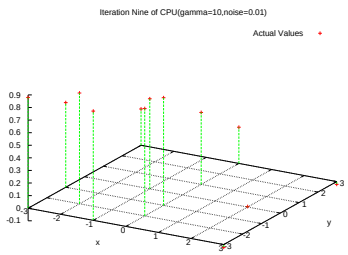
(f) Iteration vi



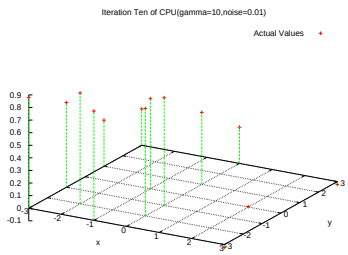
(g) Iteration vii



(h) Iteration viii



(i) Iteration ix



(j) Iteration x

Figure 3.5: Actual Values with $\gamma_{GPO} = 10$ and $\delta_{GPO} = 0.01$ in 10 iterations

enables one to predict the position and the value of the next likely candidate point even if sometimes prediction errors exist. Another difference is that GPO will stop when further performance evaluations are likely to be irrelevant.

Of course, if there is no stopping criterion to meet, GPO will search all points of the search space, run all iterations to collect all the performance scores, and find out the best performance to build the final model. Nothing is gained compared to brute-force Grid Search. Thus, the stopping criterion is significant. Note also that some poor performance scores generally exist in the search space and even sometimes can affect the final performance. It is unnecessary to waste time at those points of the space and GPO needs to ignore them through its stopping criterion. The ideal is to search as few points as possible among all points without influencing the final performance.

As mentioned before, the user can specify the parameters of the base algorithm's search as well as the step size. Consider the "CPU" dataset, for instance. We again choose the Gaussian process algorithm as the base classifier together with the RBF kernel, so the parameters to be found are δ in the Gaussian process and γ in the RBF kernel. Assume the minimum and the maximum value for γ on the x axis as well as for δ on the y axis are -5 and 5 respectively (as exponents) and the step size between neighbouring points is 1. Then the grid includes 121 pairs of points altogether. Thus, the program will perform 121 iterations when not using a stopping criterion. This process is clearly inflexible in general. Suppose the dataset we use has thousands of instances or plenty of attributes, or the search space is very huge or the step size is quite fine: it will cost a large amount of time to run the optimization process and the number of performance scores for all points will be enormous. The most important point is that most of the results will be useless.

Let us consider some further results and a concrete stopping criterion. To obtain these results, we first collected the initial training data with the parameter points corresponding to the four corners of the search space. New data was added into the training dataset after every iteration. As before, these training data points are used for building the GP model with specified δ_{GPO} value and specified γ_{GPO} value. In this case, we again set the γ_{GPO} value to 10 and the δ_{GPO} value to 0.01.

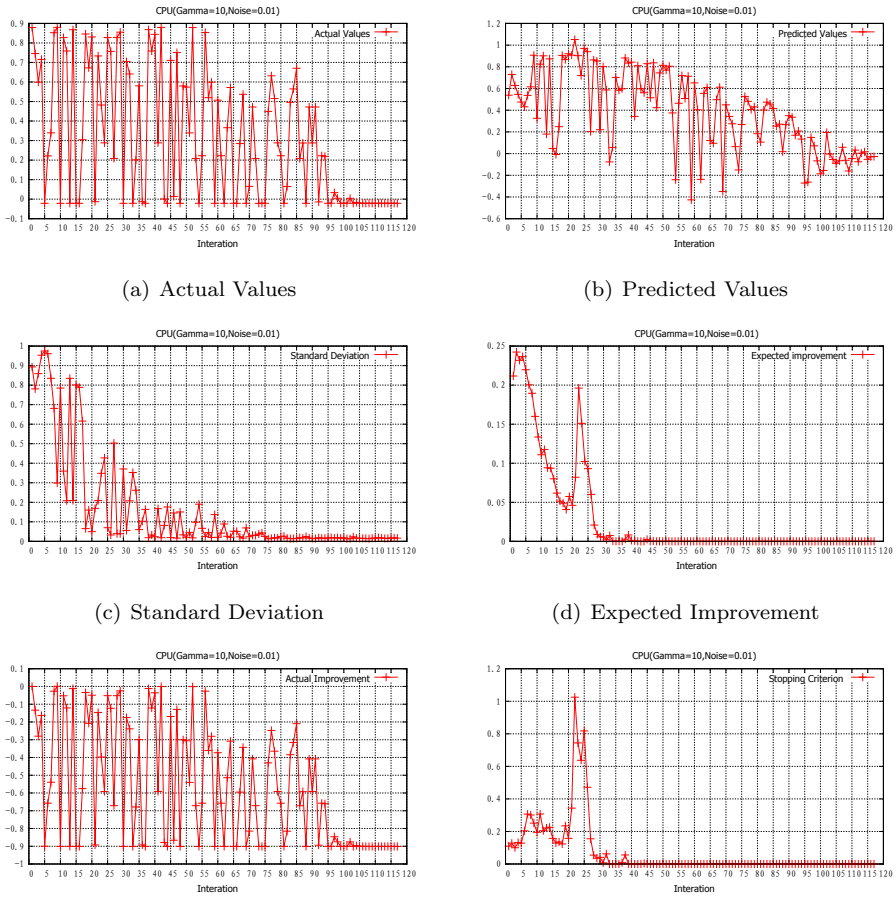
Figure 3.6 shows six tendency curves observed during 121 iterations based on the GP model with $\gamma_{GPO} = 10$ and $\delta_{GPO} = 0.01$ on the "CPU" dataset. These curves show actual value (a), predicted value (b), standard deviation (c), expected improvement (d), actual improvement (e) and stopping criterion (f). The actual values are observed performance scores of the chosen point. They have been evaluated and cross-validated, and are treated as target values of the training dataset for the GP model as well. The points in the grid with unknown values are able to receive a prediction through this model but these predictions are obviously not as accurate as the true values due to unavoidable uncertainty. The standard deviation, in short, is the uncertainty at some point. The expected improvement has been discussed in Section 3.2. Here, the expected improvement is the maximum expected improvement of each iteration. Compared with expected improvement, the difference between the current actual value and the "best" actual previous value is defined as the actual improvement. The stopping criterion we use is calculated according to the equation in [13] (further details on the full criterion are given below):

$$stopping \quad criterion \quad = \quad \frac{(max(EI) - avg(EI))^2}{sd(EI)} < 0.1 \tag{3.3}$$

where EI is the abbreviation of expected improvement, $max(EI)$, $avg(EI)$ and $sd(EI)$ is the maximum expected improvement, the average expected improvement and the standard deviation of expected improvement respectively, based on EI scores for all grid locations from the current iteration. This criterion is designed to detect when the average expected improvement $avg(EI)$ is close to the current maximum expected improvement $max(EI)$ with a low standard deviation $sd(EI)$. The optimization procedure can stop there. A small difference between the maximum expected improvement and the average indicates that the search may have already found all valuable points, and a low standard deviation means that the uncertainty becomes smaller and smaller.

Figure 3.7 shows the 6 measure charts with the same γ_{GPO} and δ_{GPO} values on the "concrete" dataset. The plots have many differences from those in Figure 3.6 in terms of absolute values but are quite similar in tendency.

What we want is to identify the best performance among actual values, and we can simply see that the best actual values appear in the early iterations in both Figure 3.6 and Figure 3.7. In both cases, the evaluation (actual values) is



(a) Actual Values

(b) Predicted Values

(c) Standard Deviation

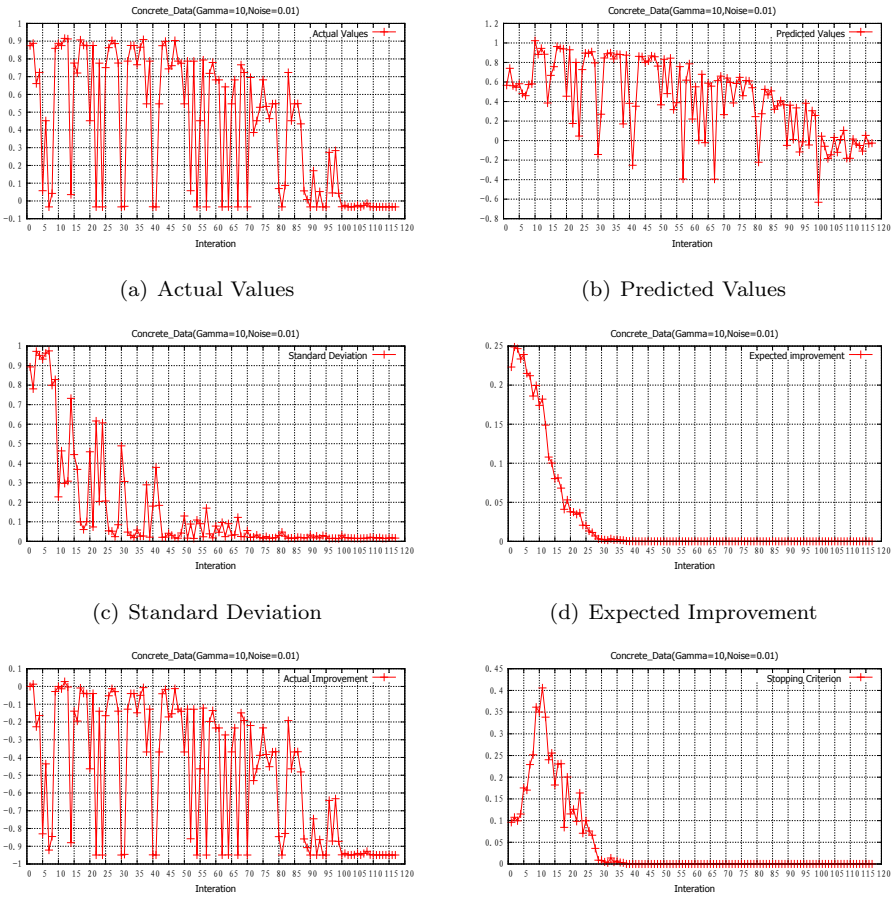
(d) Expected Improvement

(e) Actual Improvement

(f) Stopping Criterion

Figure 3.6: Tendency charts for chosen points when exhausting the full parameter space with $\gamma_{GPO} = 10$ and $\delta_{GPO} = 0.01$ based on the "CPU" dataset

based on cross-validated correlation coefficient (because we are dealing with a regression problem). The standard deviation decreases gradually though some small fluctuations can be observed and it is close to 0 in the last few iterations. Ideally, we stop in a way such that we not only include the "best" point, but also search fewer points. The stopping criterion in [13] is applied here: we stop if the current best value is smaller or equal to the old one, the squared difference between the maximal expected improvement and the average expected improvement is less than 0.1 of the standard deviation of the expected improvement (i.e. Equation 3.3 has a value smaller than 0.1), and the current best value did not change during the last 10 iterations in order to prevent long searches.



(a) Actual Values

(b) Predicted Values

(c) Standard Deviation

(d) Expected Improvement

(e) Actual Improvement

(f) Stopping Criterion

Figure 3.7: Tendency charts for chosen points when exhausting the full parameter space with $\gamma_{GPO} = 10$ and $\delta_{GPO} = 0.01$ based on the "concrete" dataset

As we can see from Figure 3.6 and 3.7, this criterion applied to terminate GPO search works very well for the "CPU" and "concrete" datasets. Note again that the γ_{GPO} quantity for the GP model was set to 10 and the δ_{GPO} quantity for the RBF kernel was set to 0.01. Empirically, other values considered result in less desirable behaviour, in particular for the stopping criterion. In the next chapter, this criterion, with these two parameter values, will be applied in both regression and classification datasets to test the effectiveness of GPO in the Experimenter interface of WEKA.

3.6 Summary

The goal of Gaussian Process Optimization is to pick the globally optimal point with the highest expected improvement. The possible points are chosen from the parameter range specified by the user's empirical experience. The most common method to pick the most likely point is a trial-and-error procedure, that is, we try each point in the grid and get the lowest error point, as was described in Section 3.1. In this thesis, we adopt the Gaussian Process Optimization method to search for this optimal point. It is also important to perform the appropriate parameter settings for the GP model, which was elaborated on in Section 3.5. Apart from the above, we have considered the stopping criterion that is used to determine when the GPO process stops. The progress observed after each iteration was displayed in figures in order to show details clearly, and thus to decide when it is appropriate to stop. Once the stopping criterion applies, the evaluation based on 2-fold cross-validation will finish and the "best" point from this first stage search is identified. The stopping criterion can prevent the searching and evaluating processes going further, as only part of the range of parameter values is worthy of analysis. In the next stage, the search will perform 10-fold cross-validation around this "best" point of the above stage until no better point is found.

Chapter 4

Experimental Results

In the previous chapter, we discussed the problem of parameter optimization based on the Gaussian processes framework, which generalizes characteristics from previous samples and provides predictions for future samples as well. We have experimentally determined an appropriate pair of parameters $(\delta_{GPO}, \gamma_{GPO})$ for the Gaussian processes model, compared with other pairs of parameters. In addition, the stopping criterion used to prevent the search from going further than necessary has shown good behaviour in preliminary experiments. However, there is so far not sufficient evidence that Gaussian Process Optimization works well in practice when tuning machine learning algorithms. Hence, it is important to perform experiments showing the effects of this optimization method for some widely-used algorithms.

On the one hand, we will use this optimization algorithm with classification learning, such as decision tree learning, i.e. $C4.5$, Support Vector Machines, and Boosting. On the other hand, we will apply the optimization algorithm with Gaussian process regression to test the effectiveness of the method on regression problems. In this chapter, Sections 4.1 and 4.2 will show the results of parameter optimization in regression and classification problems respectively using the stopping criterion whose behaviour has been examined previously. The quality of the Gaussian Process Optimization method will be determined through statistical tests. All the outcomes of the experiments are based on implementations in WEKA.

4.1 Regression

Regression problems refer to cases where the predictive outputs are continuous. There are many evaluation measures to estimate the performance on the same regression problem for various algorithms. Different measurements will bring different performance estimates. Measures for numeric prediction include mean-squared error, mean absolute error, relative absolute error, relative squared error and correlation coefficient. All these measurements are concerned with the relationship between predicted values for the test instances and the actual values. Every evaluation measure has its own disadvantages and advantages. The preferred performance measure for the learning algorithms differs according to the user's viewpoint [1]. The decision for choosing a measure depends on the objective of a particular implementation.

In the following, we will use the correlation coefficient as the primary measure in numeric prediction problems. The correlation coefficient measures the statistical correlation between the actual values and the predicted values. In general, the value of the correlation coefficient ranges from -1 for no correlation to 1 for correlated perfectly. In other words, an algorithm with a higher correlation coefficient is better than one with a lower one when comparing two performance scores. It is thus different from the other error measures, because a larger value of the correlation coefficient indicates good performance, while good performance for the latter ones is indicated by smaller values.

Apart from the above, another standard to estimate efficiency is the use of time. Researchers are not willing to spend a long time to wait for outcomes of experiments. Therefore, we will compare the run time for different optimization algorithms on the same regression task. As a large amount of time is spent on cross-validation, we will also consider the number of cross-validation runs performed.

4.1.1 Numeric Prediction Based on Gaussian Process Regression

Gaussian process regression is a widely used regression method. It is important to note that the Gaussian process model being optimized in the following

Table 4.1: The search configuration of GPO and Grid Search in regression

Common points in GPO and Grid Search	Base algorithm: Gaussian process function Search parameters: γ of RBF kernel function in the classifier treated as x_1 and the Gaussian noise δ of the base algorithm in the classifier treated as x_2 Search range: I$\in \{-5, -4, \ldots, 5\}$ Search step: 1 Search base: 10 Search expression: 10^I Search route: by columns
Difference in GPO and Grid Search	GPO algorithm builds GP model for predictive distribution, the parameters of which are the Gaussian noise δ and the width γ of the RBF kernel function. We specify the γ value as 10 and the noise level δ as 0.01.

is different from the GP model used inside the GPO process. We will first compare GPO and Grid Search when optimizing a Gaussian process model. The configuration for both GPO and Grid search is set in advance, see Table 4.1. To make it easier to understand, we list the common points as well as the differences of these two algorithms separately. It is clear that the only distinct difference between GPO and Grid Search is the use of the GP model for prediction. That is GPO uses the Gaussian processes distribution to evaluate all the possible points in the grid before determining the next sample point.

First, we will compare the performance in terms of correlation coefficient when optimizing it with both GPO and Grid Search. The performance on a single dataset cannot determine the quality of the algorithm. Hence, we use a large collection (see Table 4.2). We use cross-validation estimates to determine the mean performance for each dataset. Based on this we can determine whether one algorithm significantly outperforms the other algorithm. Because of a matched pair of results for each dataset for both learning algorithms, the statistical test we use is known as the paired t-test. More precisely, taking 10-fold cross-validation for example, a set of outcomes x_i will be obtained using one learning algorithm, and another set of outcomes y_i will be obtained by the other learning algorithm on the same dataset. The mean of the first set of outcomes x_i and the mean of the second set of outcomes y_i can be calculated. Comparing the two values of the mean, along with corresponding standard deviations, we will know whether the first learning algorithm is significantly different from the

other one in terms of performance. The results shown in Table 4.2 were obtained after running 10-fold cross-validation repeated 10 times. Thus, the value of every entry in Table 4.2 is the average of 100 values.

The table shows that the correlation coefficient is similar for all datasets for both GPO and Grid Search. There is no significantly greater, or significantly lower result between GPO and Grid Search, even though the value of GPO in some cases is a little bit higher than that of Grid Search, and the value of Grid Search is a little better than that of GPO in some other ones. It is encouraging that there are no statistically significant changes between these two algorithms because Grid Search does a brute-force search in the range of the grid. It can be observed that GPO performs equally well, and as we will see it only does relatively few searches to find the best performance.

Let us now consider computational complexity. The number of cross-validation runs is an important measure to test the GPO and Grid Search algorithms. Only points selected based on expected improvement need to be cross-validated in the GPO algorithm, whereas Grid Search will use cross-validation for every point in the whole parameter search space. For the GPO algorithm as well as the Grid Search algorithm, there are two stages of cross-validation. The first stage is based on 2-fold cross-validation, and the second stage is based on 10-fold cross-validation. The initial "best" point is found using 2-fold cross-validation. However, based on 10-fold cross-validation after the first stage, this point may fall into a sub-optimal area: there is the possibility that there are better points around the point found in the first stage. Thus, it is essential to run 10-fold cross-validation on surrounding points to obtain more accurate estimates. Overall, dividing the cross-validation into two stages is done for two main reasons: saving time and avoiding sub-optimum parameter settings.

Table 4.3 compares how many 2-fold cross-validation runs both GPO and Grid Search perform. The hollow circles mean that the values of the second column exhibit a statistically significant improvement compared to the values of the third column. The GPO algorithm outperforms the Grid Search algorithm on the 2-fold cross-validation. Considering both Table 4.2 and Table 4.3, we find the correlation coefficient does not decrease significantly with fewer 2-fold cross-validation runs. Take the "basketball" dataset for example. The GPO

Table 4.2: Correlation Coefficient for GPO and Grid Search

Dataset	GPO	Grid Search
auto93.names	0.8443 ± 0.1227	0.8445 ± 0.1228
autoHorse.names	0.9494 ± 0.0968	0.9501 ± 0.0969
autoMpg.names	0.9315 ± 0.0253	0.9320 ± 0.0247
autoPrice.names	0.9030 ± 0.0757	0.9078 ± 0.0738
baskball	0.5778 ± 0.2282	0.5760 ± 0.2259
bodyfat.names	0.9838 ± 0.0295	0.9834 ± 0.0293
bolts	0.8844 ± 0.2438	0.8892 ± 0.2444
breastTumor	0.2234 ± 0.1864	0.2796 ± 0.1684
cholesterol	0.1528 ± 0.1818	0.1701 ± 0.1763
cleveland	0.7250 ± 0.0834	0.7222 ± 0.0838
cloud	0.9210 ± 0.0717	0.9196 ± 0.0779
cpu	0.9965 ± 0.0064	0.9966 ± 0.0064
detroit	0.1800 ± 0.5199	0.2200 ± 0.5041
echoMonths	0.6958 ± 0.1408	0.6970 ± 0.1409
elusage	0.8483 ± 0.1819	0.8432 ± 0.1814
fishcatch	0.9896 ± 0.0077	0.9899 ± 0.0080
fruitfly	-0.1047 ± 0.2431	-0.0869 ± 0.2716
gascons	0.9391 ± 0.2868	0.9367 ± 0.2868
housing	0.9339 ± 0.0324	0.9405 ± 0.0299
hungarian	0.7137 ± 0.0995	0.7170 ± 0.0969
longley	0.5600 ± 0.5379	0.5800 ± 0.5160
lowbwt	0.7845 ± 0.0770	0.7835 ± 0.0777
mbagrade	0.3817 ± 0.4775	0.3920 ± 0.4804
meta	0.3797 ± 0.2337	0.3752 ± 0.2268
pbc	0.5931 ± 0.1020	0.5917 ± 0.1035
pharynx	0.6826 ± 0.1116	0.6898 ± 0.1105
pollution	0.7283 ± 0.2596	0.7367 ± 0.2569
pwLinear	0.9146 ± 0.0367	0.9167 ± 0.0372
quake	0.0818 ± 0.0899	0.0735 ± 0.0887
schlvote	0.1655 ± 0.5888	0.1862 ± 0.6143
sensory	0.5116 ± 0.0942	0.5149 ± 0.0921
servo	0.9173 ± 0.0514	0.9183 ± 0.0500
sleep	0.6102 ± 0.3872	0.6203 ± 0.3809
strike	0.5293 ± 0.1917	0.5299 ± 0.1984
veteran	0.3959 ± 0.2587	0.3828 ± 0.2566
vineyard	0.7242 ± 0.2707	0.7134 ± 0.3097

◦, • statistically significant improvement or degradation

algorithm performs just 18 2-fold cross-validations on average to get a slightly better performance (an average of 0.5778 in correlation coefficient). The performance is 0.5760 for Grid Search although it runs 121 2-fold cross-validations.

Once the initial "best" point has been decided using 2-fold cross-validation, both GPO and Grid Search will perform 10-fold cross-validation around the neighbours of this "best" point in order to find a potentially better one. In both algorithms, the search does not stop until there is no improvement or the searching point is on the boundary of the parameter space. As a result, both usu-

Table 4.3: Number of 2-Fold Cross-validation Runs in GPO and Grid Search

Dataset	GPO	Grid Search
auto93.names	22.8800 ± 8.0293	121.0±0.0 ∘
autoHorse.names	20.0200± 6.6286	121.0±0.0 ∘
autoMpg.names	17.6600± 7.3899	121.0±0.0 ∘
autoPrice.names	19.8000± 7.7277	121.0±0.0 ∘
baskball	17.4800± 7.4488	121.0±0.0 ∘
bodyfat.names	16.6200± 5.1006	121.0±0.0 ∘
bolts	16.4200± 5.0795	121.0±0.0 ∘
breastTumor	18.1900± 7.9629	121.0±0.0 ∘
cholesterol	16.6000± 6.9486	121.0±0.0 ∘
cleveland	15.6300± 3.8023	121.0±0.0 ∘
cloud	14.4300± 4.3491	121.0±0.0 ∘
cpu	15.0800± 4.4827	121.0±0.0 ∘
detroit	25.3500±10.0054	121.0±0.0 ∘
echoMonths	16.6600± 5.5473	121.0±0.0 ∘
elusage	16.4800± 6.0427	121.0±0.0 ∘
fishcatch	18.4600± 7.0316	121.0±0.0 ∘
fruitfly	13.3100± 4.5676	121.0±0.0 ∘
gascons	20.5800± 8.4736	121.0±0.0 ∘
housing	16.8700± 7.8504	121.0±0.0 ∘
hungarian	23.8900± 8.5527	121.0±0.0 ∘
longley	18.7200± 7.1096	121.0±0.0 ∘
lowbwt	16.0700± 4.3791	121.0±0.0 ∘
mbagrade	13.7700± 4.4966	121.0±0.0 ∘
meta	17.3900± 7.0980	121.0±0.0 ∘
pbc	20.1400± 6.4527	121.0±0.0 ∘
pharynx	17.3700± 6.5685	121.0±0.0 ∘
pollution	16.8500± 6.5310	121.0±0.0 ∘
pwLinear	22.2000± 9.4345	121.0±0.0 ∘
quake	14.9300± 5.0817	121.0±0.0 ∘
schlvote	17.8600± 7.4332	121.0±0.0 ∘
sensory	17.2500± 8.0156	121.0±0.0 ∘
servo	28.0900±11.4372	121.0±0.0 ∘
sleep	19.0600± 7.3551	121.0±0.0 ∘
strike	17.0300± 6.5635	121.0±0.0 ∘
veteran	18.6500± 8.7575	121.0±0.0 ∘
vineyard	15.3100± 5.7361	121.0±0.0 ∘

∘, • statistically significant improvement or degradation

ally perform several 10-fold cross-validation runs to get the final performance, as illustrated in Table 4.4.

Normally, 10-fold cross-validation tends to cost more running time and acquires more accurate performance estimates than 2-fold cross-validation. In this case, the grid has 8 neighbours for each central point. For most datasets here, the number of 10-fold cross-validation runs in GPO is smaller than that of Grid Search, which is another piece of evidence that GPO exceeds Grid Search in terms of computational efficiency.

Table 4.4: Number of 10-Fold Cross-validation Runs in GPO and Grid Search

Dataset	GPO	Grid Search
auto93.names	12.1500 ± 4.8770	12.3500 ±3.0563
autoHorse.names	8.4200 ± 5.8122	10.5500 ±3.7047
autoMpg.names	11.1100 ± 8.2949	11.1900 ±4.3986
autoPrice.names	13.0000 ± 7.3278	14.0300 ±5.6933
baskball	10.0200 ±10.1812	10.4700 ±8.6064
bodyfat.names	7.0100 ± 6.5590	9.9200 ±3.2989
bolts	3.2300 ± 6.2746	5.7000 ±8.2603
breastTumor	3.7100 ± 5.4054	7.1800 ±6.0759
cholesterol	5.4600 ± 7.4025	8.2400 ±6.0472
cleveland	3.8800 ± 5.6053	8.3700 ±5.2081 ∘
cloud	2.9300 ± 5.3791	4.6600 ±6.2137
cpu	2.0500 ± 4.4843	2.5400 ±5.0701
detroit	10.6400 ± 6.5095	12.8300 ±4.7738
echoMonths	3.5100 ± 5.2924	8.1800 ±4.9937 ∘
elusage	4.4500 ± 6.8230	5.3800 ±6.9293
fishcatch	10.5700 ± 6.9939	9.7600 ±4.8244
fruitfly	3.0000 ± 5.9645	5.8900 ±6.7643
gascons	9.4200 ± 7.5387	10.3900 ±4.5369
housing	8.9900 ± 9.4479	10.2300 ±6.0734
hungarian	9.4500 ± 5.8090	11.6700 ±4.3066
longley	6.9400 ± 6.5672	9.9200 ±6.5082
lowbwt	2.4700 ± 4.8021	4.0800 ±5.3817
mbagrade	2.8700 ± 6.3575	5.9500 ±9.7041
meta	4.2300 ± 6.5256	7.7400 ±6.4895
pbc	7.8100 ± 4.6898	8.3600 ±3.2178
pharynx	4.9500 ± 5.8021	3.9100 ±4.9054
pollution	6.2700 ± 5.9592	10.2000 ±3.8271
pwLinear	12.8300 ± 6.1760	12.7600 ±4.4859
quake	1.9200 ± 4.9781	3.3400 ±5.0836
schlvote	5.5800 ± 8.3414	8.6200 ±7.9096
sensory	7.8800 ± 7.8138	8.2100 ±6.0491
servo	11.3100 ± 4.9292	10.3600 ±2.7470
sleep	9.7500 ± 7.6599	11.1800 ±5.4539
strike	6.3900 ± 7.2361	10.7800 ±5.1081
veteran	9.3600 ± 9.5182	11.6600 ±7.5895
vineyard	5.3600 ± 7.1978	7.9400 ±7.9224

∘, • statistically significant improvement or degradation

The different amount of cross-validation and the GP model used for the GPO algorithm are the main reasons for the difference in the training time. Table 4.5 shows the training time for the 36 datasets for both GPO and Grid Search. Grid Search spends twice as much training time on evaluating and performing the search than GPO for most datasets, even four or five times as much for some datasets, i.e. the "cleveland" dataset and the "breastTumor" dataset. It is seen that GPO yields an obvious improvement in running time when compared with Grid Search. GPO enables us to consume much less time

Table 4.5: Training Time of GPO and Grid Search

Dataset	GPO		Grid Search		
auto93.names	2.0776±	0.5862	3.7351 ±	0.4251	∘
autoHorse.names	2.9111±	1.4850	6.6518 ±	0.8424	∘
autoMpg.names	10.6744±	6.9722	22.7175 ±	3.6393	∘
autoPrice.names	2.5259±	1.1093	4.7633 ±	0.8689	∘
baskball	1.4741±	1.0428	3.1960 ±	0.8363	∘
bodyfat.names	2.8162±	1.9611	7.6328 ±	0.9593	∘
bolts	0.7295±	0.5549	2.4083 ±	0.6702	∘
breastTumor	2.6122±	2.3234	9.0870 ±	2.3489	∘
cholesterol	3.5035±	3.2076	10.4319 ±	2.4815	∘
cleveland	2.5867±	2.4628	10.1145 ±	2.1627	∘
cloud	0.7651±	0.5988	2.7429 ±	0.6859	∘
cpu	1.1469±	1.0599	4.1356 ±	1.1557	∘
detroit	1.4964±	0.6292	2.9468 ±	0.5101	∘
echoMonths	0.9556±	0.7170	3.3770 ±	0.5942	∘
elusage	0.8495±	0.6595	2.4638 ±	0.5842	∘
fishcatch	2.1211±	1.0380	4.0795 ±	0.6891	∘
fruitfly	0.8277±	0.7609	3.1792 ±	0.8874	∘
gascons	1.2890±	0.6686	2.7340 ±	0.4269	∘
housing	15.3489±	14.0341	29.2449 ±	8.6990	∘
hungarian	5.0440±	2.5017	10.8289 ±	1.7025	∘
longley	1.0460±	0.6123	2.7156 ±	0.5761	∘
lowbwt	1.1074±	0.9518	4.4642 ±	1.0368	∘
mbagrade	0.6518±	0.5519	2.5233 ±	0.7918	∘
meta	11.6704±	11.9962	42.3691 ±	11.8745	∘
pbc	9.2670±	4.4986	21.6678 ±	2.9820	∘
pharynx	2.7256±	1.8770	7.3625 ±	1.5262	∘
pollution	1.0111±	0.5974	2.8566 ±	0.3346	∘
pwLinear	3.2743±	1.2612	5.9724 ±	0.8656	∘
quake	811.7735±	1595.5955	2310.9863 ±	1985.8234	
schlvote	0.9707±	0.7544	2.5678 ±	0.6119	∘
sensory	20.1986±	17.0838	34.0367 ±	12.8195	∘
servo	2.7390±	0.8268	4.4110 ±	0.4169	∘
sleep	1.3778±	0.7495	2.8543 ±	0.4362	∘
strike	20.7891±	19.7951	47.6723 ±	13.3069	∘
veteran	1.8575±	1.3797	4.0907 ±	1.0864	∘
vineyard	0.9049±	0.7119	2.5458 ±	0.6245	∘

∘, • statistically significant improvement or degradation

but with comparative performance: Grid Search has to do the complete trial process.

Overall, we discussed four measures above, including correlation coefficient performance, number of 2-fold cross-validation runs, number of 10-fold cross-validation runs and training time, comparing the GPO algorithm with the Grid Search algorithm on regression problems. GPO is generally better in training time and number of 2-fold cross-validation runs with similar correlation coeffi-

cients and this does not vary much in all datasets used in this test. At the same time, most cases also exhibit relatively fewer 10-folds cross-validations for GPO than for Grid Search. However, GPO does not work quite as well in some cases, such as on the "detroit" dataset due to the relatively lower correlation coefficient and higher number of 2-fold cross-validation runs. Some possible reasons are as follows:

1: The stopping criterion used here is not perfect and may not consider all the best points. Some of them may be missed. As the result of this stopping criterion, we can not search all the points as Grid Search does. Some valuable and important points with lower expected improvement and higher uncertainty may be ignored by the GPO algorithm.

2: In the GPO algorithm, we use a set of corner points with their corresponding performance scores as an initial training dataset for the GP model, which decides the beginning tendency for the future unknown points. If the initial points have not been chosen suitably, it will result in some bias in subsequent choices of points.

4.1.2 Comparing GPO and Random Search

We found that the GPO algorithm generally performs better than Grid Search through the above analysis. We also found the weakness of Grid Search is the usage of time because of its brute-force nature. Hence, we propose an experiment to compare the efficiency of GPO to Random Search. Random search is a simple search method that searches points randomly and does not require as much time as Grid Search. Before we apply random search as the search method, we set the number of random points first. More specifically, the random numbers are the points in the grid generated by the random number generator. If the number of random numbers is set to m, then we randomly choose m points from the grid. In the following experiments, we will use five values for m, from 5 to 25 with step size 5. For every experiment, a set of outcomes will be obtained and compared with the GPO algorithm. We adopt the same evaluation measures as before, namely correlation coefficient, number of cross-validation runs and training time.

Table 4.6: Correlation Coefficient of GPO and Random Search

Dataset	GPO	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
auto93.names	0.8458±0.12	0.8644±0.11	0.8644±0.11	0.8451±0.12	0.8451±0.12	0.8451±0.12
autoHorse.names	0.9581±0.03	0.9509±0.08	0.9509±0.08	0.9517±0.08	0.9592±0.02	0.9584±0.03
autoMpg.names	0.9320±0.02	0.9325±0.02	0.9325±0.02	0.9325±0.02	0.9325±0.02	0.9328±0.02
autoPrice.names	0.9065±0.07	0.8928±0.07	0.8928±0.07	0.9063±0.07	0.9063±0.07	0.9083±0.07
baskball	0.5733±0.23	0.5805±0.22	0.5805±0.22	0.5825±0.22	0.5738±0.23	0.5849±0.22
bodyfat.names	0.9835±0.03	0.9838±0.03	0.9838±0.03	0.9829±0.03	0.9829±0.03	0.9825±0.03
bolts	0.8982±0.22	0.8833±0.27	0.8833±0.27	0.9329±0.19	0.9261±0.20	0.9261±0.20
breastTumor	0.2770±0.17	0.2688±0.16	0.2667±0.16	0.2656±0.17	0.2783±0.18	0.2060±0.21
cholesterol	0.1666±0.17	0.1602±0.18	0.1657±0.18	0.1686±0.18	0.1782±0.17	0.1189±0.19
cleveland	0.7237±0.08	0.7203±0.09	0.7238±0.08	0.7243±0.08	0.7243±0.08	0.7161±0.09
cloud	0.9211±0.07	0.9218±0.07	0.9218±0.07	0.9210±0.07	0.9210±0.07	0.9146±0.08
cpu	0.9968±0.01	0.9970±0.01	0.9970±0.01	0.9973±0.01	0.9971±0.01	0.9971±0.01
detroit	0.2200±0.50	0.2200±0.50	0.2200±0.50	0.2400±0.49	0.2400±0.49	0.2400±0.49
echoMonths	0.6977±0.14	0.6823±0.15	0.6776±0.16	0.6954±0.15	0.6942±0.15	0.6977±0.14
elusage	0.8494±0.18	0.8565±0.18	0.8583±0.18	0.8441±0.19	0.8569±0.17	0.8494±0.17
fishcatch	0.9899±0.01	0.9873±0.01	0.9873±0.01	0.9874±0.01	0.9896±0.01	0.9901±0.01
fruitfly	-0.0982±0.27	-0.1254±0.26	-0.1232±0.26	-0.1445±0.23	-0.1168±0.23	-0.0521±0.23
gascons	0.9168±0.35	0.9361±0.29	0.9361±0.29	0.9339±0.29	0.9354±0.29	0.9385±0.29
housing	0.9371±0.03	0.9326±0.03	0.9326±0.03	0.9326±0.03	0.9400±0.03	0.9456±0.03
hungarian	0.7171±0.10	0.7178±0.10	0.7168±0.10	0.7175±0.10	0.7175±0.10	0.7175±0.10
longley	0.5800±0.52	0.5600±0.54	0.5600±0.54	0.5800±0.52	0.5800±0.52	0.6000±0.49
lowbwt	0.7845±0.08	0.7814±0.08	0.7814±0.08	0.7848±0.08	0.7848±0.08	0.7805±0.08
mbagrade	0.3821±0.49	0.4108±0.46	0.4108±0.46	0.4079±0.46	0.4079±0.46	0.3890±0.48
meta	0.3681±0.23	0.3923±0.21	0.3923±0.21	0.3917±0.22	0.3754±0.22	0.3454±0.25
pbc	0.5924±0.10	0.5939±0.10	0.5939±0.10	0.5939±0.10	0.5939±0.10	0.5931±0.10
pharynx	0.6837±0.12	0.6046±0.13 •	0.6053±0.13 •	0.6909±0.11	0.6909±0.11	0.6909±0.11
pollution	0.7357±0.25	0.7142±0.27	0.7142±0.27	0.7306±0.26	0.7577±0.24	0.7336±0.27
pwLinear	0.9160±0.04	0.9165±0.04	0.9165±0.04	0.9173±0.04	0.9173±0.04	0.9173±0.04
quake	0.0806±0.09	0.0600±0.09	0.0598±0.09	0.0518±0.09	0.0493±0.09	0.0690±0.07
schlvote	0.1970±0.59	0.1799±0.60	0.1799±0.60	0.1501±0.60	0.2136±0.57	0.1282±0.59
sensory	0.5144±0.09	0.5135±0.09	0.5135±0.09	0.5154±0.09	0.5154±0.09	0.5027±0.10
servo	0.9177±0.05	0.9179±0.05	0.9179±0.05	0.9170±0.05	0.9169±0.05	0.9161±0.05
sleep	0.5952±0.39	0.6062±0.41	0.6062±0.41	0.6014±0.41	0.6007±0.41	0.5627±0.42
strike	0.5292±0.20	0.5264±0.20	0.5264±0.20	0.5286±0.20	0.5266±0.20	0.5316±0.20
veteran	0.3948±0.25	0.3853±0.26	0.3853±0.26	0.3882±0.26	0.3944±0.26	0.3651±0.27
vineyard	0.7051±0.30	0.7047±0.31	0.7047±0.31	0.7154±0.31	0.6969±0.31	0.7118±0.30
Average	0.6497	0.6453	0.6455	0.6495	0.6529	0.6460

○, • statistically significant improvement or degradation

Table 4.6 shows the performance of GPO and Random Search based on correlation coefficient. There is no statistically significant improvement after changing from Grid Search to the Random Search. However, an interesting phenomenon occurs for the particular dataset "pharynx". The GPO algorithm outperforms random search statistically significantly when 25 and 20 random grid locations are used respectively. However, this phenomenon disappears after decreasing the number of random grid locations. The reason is that the points found in these two situations above are located on the border of the grid, and the procedure prevents the search process going further.

The number of random grid locations is equal to the number of 2-fold cross-validation runs in Random Search. According to Table 4.7, the average number of 2-fold cross-validation runs is close to 25 using the GPO algorithm, where the biggest value is around 30 and the smallest is about 19. As a result, the GPO algorithm shows a statistically significant degradation with decreasing random grid locations compared with the Random Search method when considering 2-fold cross-validation runs.

In contrast to the results for 2-fold cross-validation, the smaller numbers of random grid locations lead to larger numbers of 10-fold cross-validations. The 10-fold cross-validation runs are based on the 2-fold cross-validation ones, and the "best" point found by 2-fold cross-validation is likely not to be the optimal point when few random locations are explored. The goal of the 10-fold cross-validation is to visit the nearby points and try to find a point with better correlation coefficient. If too few 2-fold cross-validations are performed, the point in the first stage is not good enough, and the search process has to run 10-fold cross-validation many times in order to find the "best" one. This is also why the number of 10-fold cross-validation runs in the GPO algorithm is far less than those for RS (15), RS (10), and RS (5).

We are mainly concerned with the training time of GPO and Random Search. According to the statistically significant improvement or degradation, the results for Random Search can be divided into two groups, see Table 4.9. Random Search with 25 and 20 random points belongs to one group while the remaining searches are in the other group. The GPO algorithm shows comparative performance when the number of random grid locations in Random

Table 4.7: Number of 2-Fold Cross-Validation Runs in GPO and Random Search

Dataset	GPO	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
auto93	23.11±6.53	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
autoHorse	21.12±3.63	25±0.0 ○	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
autoMpg	22.82±4.59	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
autoPrice	23.36±4.23	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
baskball	25.33±6.22	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
bodyfat	22.84±5.42	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
bolts	19.98±4.69	25±0.0 ○	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
breastTumor	30.59±5.02	25±0.0 •	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
cholesterol	30.38±5.37	25±0.0 •	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
cleveland	22.89±5.46	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
cloud	19.49±3.33	25±0.0 ○	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
cpu	23.37±4.41	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
detroit	21.76±6.44	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
echoMonths	22.26± 5.7	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
elusage	23.61±4.93	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
fishcatch	23.94±5.04	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
fruitfly	29.58±6.00	25±0.0 •	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
gascons	22.32±4.78	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
housing	19.71±4.56	25±0.0 ○	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
hungarian	25.27±5.09	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
longley	22.62±5.62	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
lowbwt	19.99±3.34	25±0.0 ○	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
mbagrade	26.45±5.69	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
meta	30.13±5.69	25±0.0 •	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
pbc	22.98±4.82	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
pharynx	25.52±6.85	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
pollution	24.84±5.81	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
pwLinear	20.33±3.56	25±0.0 ○	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
quake	27.89±4.28	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
schlvote	28.29±6.21	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
sensory	30.14±5.94	25±0.0 •	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
servo	21.66±5.30	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
sleep	24.53±6.15	25±0.0	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
strike	30.29±5.58	25±0.0 •	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
veteran	29.91±5.57	25±0.0 •	20±0.0 •	15±0.0 •	10±0.0 •	5±0.0 •
vineyard	21.74±5.48	25±0.0	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
Average	24.47	25.0	20.0	15.0	10.0	5.0

○, • statistically significant improvement or degradation

Search is close to the average of the number of 2-fold cross-validation runs in GPO. If the set of random grid locations is too small, the GPO algorithm will perform better than Random Search. Note that a substantial advantage of GPO in practice is that it determines an appropriate number of cross-validation runs automatically using its stopping criterion. However, the selection of points based on expected improvement does not appear necessary in most cases.

Table 4.8: Number of 10-Fold Cross-Validation Runs in GPO and Random Search

Dataset	GPO	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
auto93.names	10.7700±5.60	0.0000± 0.00 •	0.0000± 0.00 •	17.8700± 2.98 ◦	17.8700±2.98 ◦	17.8700±2.98 ◦
autoHorse.names	9.3500±5.42	1.2200± 4.27 •	1.2200± 4.27 •	14.7500± 3.87 ◦	15.2600±2.87 ◦	18.9200±2.98 ◦
autoMpg.names	12.3600±6.25	20.1400± 3.18 ◦	20.1400± 3.18 ◦	20.1400± 3.18 ◦	20.1400±3.18 ◦	16.5200±2.98
autoPrice.names	12.7400±6.80	1.9200± 5.13 •	1.9200± 5.13 •	15.5400± 5.14	16.5800±3.79	21.4600±5.38 ◦
baskball	8.5500±8.42	12.0500± 8.45	12.0500± 8.45	14.3900± 7.40	17.8900±6.50 ◦	19.6100±6.93 ◦
bodyfat.names	8.7600±4.30	4.5600± 6.51	4.5600± 6.51	11.6700± 5.13	12.7800±4.02 ◦	19.0200±2.30 ◦
bolts	4.2400±7.53	4.2800± 6.61	4.2800± 6.61	9.8300± 7.69	15.7900±2.70 ◦	18.1500±1.87 ◦
breastTumor	7.1400±5.87	1.9000± 5.43 •	2.2200± 5.57 •	10.0200± 3.72	9.4900±3.05	18.8600±7.27 ◦
cholesterol	7.9300±6.78	5.9800± 7.41	6.6100± 7.44	11.2800± 5.56	11.7600±5.49	18.9600±6.09 ◦
cleveland	7.2900±5.85	1.2800± 3.87 •	1.2800± 3.87 •	9.7800± 2.02	9.7800±2.02	24.5100±5.26 ◦
cloud	4.4100±6.11	7.4600± 5.49	7.4600± 5.49	10.8700± 3.00 ◦	10.8700±3.00 ◦	22.0500±3.48 ◦
cpu	3.4000±5.18	0.6000± 3.02	0.6000± 3.02	0.9600± 3.88	16.4600±2.32 ◦	16.9100±2.25 ◦
detroit	11.5200±6.66	1.4000± 4.39 •	1.4000± 4.39 •	14.1200± 6.84	15.4900±5.81	18.2700±5.88 ◦
echoMonths	8.2300±5.03	4.3000± 6.71	4.1500± 6.62	10.8600± 2.87	9.0000±0.00	18.7400±2.80 ◦
elusage	6.3100±7.65	4.2000± 5.03	4.2900± 5.03	9.1900± 5.44	10.8600±3.95	22.1700±5.21 ◦
fishcatch	9.7200±6.34	0.3400± 2.43 •	0.3400± 2.43 •	0.4600± 2.69 •	19.0700±4.23 ◦	17.7100±2.62 ◦
fruitfly	5.4600±6.04	3.5800± 6.51	3.5100± 6.53	8.0500± 6.71	8.3200±5.91	12.1800±5.36 ◦
gascons	11.5800±5.31	10.6800±10.02	10.6800±10.02	11.5300±10.01	19.9200±4.12 ◦	17.1400±2.11 ◦
housing	7.0500±6.33	4.9200± 6.11	4.9200± 6.11	4.9200± 6.11	18.2000±6.75 ◦	12.2800±1.26 ◦
hungarian	11.3500±3.00	18.1600± 4.28 ◦	18.8100± 2.88 ◦	19.1900± 0.99 ◦	19.1900±0.99 ◦	21.2800±1.68 ◦
longley	8.7200±6.88	5.3800± 5.25	5.3800± 5.25	11.1000± 4.64	11.7200±3.73	17.6100±3.65 ◦
lowbwt	3.5600±5.36	2.9000± 5.07	2.9000± 5.07	9.5400± 2.30 ◦	9.5400±2.30 ◦	20.7700±4.19 ◦
mbagrade	5.3700±9.49	8.0500± 6.96	8.2900± 7.09	10.3700± 6.61	12.4700±4.83 ◦	21.9600±6.74 ◦
meta	6.9700±6.82	7.1000± 6.46	7.1000± 6.46	11.1900± 4.06 ◦	11.7100±4.36	17.7900±3.73 ◦
pbc	8.9600±3.45	5.4300± 6.94	5.4300± 6.94	13.2600± 2.70 ◦	13.2600±2.70 ◦	21.2400±2.17 ◦
pharynx	4.7600±5.60	0.0000± 0.00 •	0.0000± 0.00 •	17.9600± 6.46 ◦	17.9600±6.46 ◦	22.5500±0.70 ◦
pollution	8.7100±4.97	4.0700± 6.60	4.0700± 6.60	14.0600± 5.49 ◦	16.0900±5.84 ◦	23.5800±2.25 ◦
pwLinear	12.6700±4.65	18.1200± 3.84 ◦	18.1200± 3.84 ◦	18.3400± 3.39 ◦	18.5000±3.37 ◦	20.6600±3.51 ◦
quake	3.5700±5.89	6.8300± 7.10	6.8000± 7.03	9.6100± 7.10 ◦	8.7100±6.81	10.3900±2.86 ◦
schlvote	7.3700±7.70	8.6800± 7.65	8.6800± 7.65	11.8000± 6.72	12.7500±5.68	15.9300±7.35 ◦
sensory	10.3200±5.80	18.2300± 4.17 ◦	18.2300± 4.17 ◦	18.8800± 2.73 ◦	18.8800±2.73 ◦	20.2700±2.03 ◦
servo	10.3500±3.30	3.1300± 6.96 •	3.1300± 6.96 •	16.7000± 2.86 ◦	16.7800±2.90 ◦	16.4100±2.47 ◦
sleep	10.8700±5.78	8.8300± 9.19	8.7900± 9.13	15.7900± 5.06 ◦	14.5500±5.09	21.3600±5.49 ◦
strike	10.8300±5.48	9.2600± 6.39	9.2600± 6.39	12.2300± 4.22	12.6900±6.18	17.1300±2.77 ◦
veteran	11.6700±8.61	13.4100± 6.69	13.4100± 6.69	14.2200± 5.80	16.1400±6.10	16.6400±7.45
vineyard	6.3100±7.81	2.2600± 6.00	2.2600± 6.00	4.8300± 7.71	16.0100±4.43 ◦	18.4400±6.21 ◦
Average	8.3103	6.4069	6.4525	12.0917	14.5133	18.7594

◦, • statistically significant improvement or degradation

Table 4.9: Training Time of GPO and Random Search

Dataset	GPO		RS(25)		RS(20)		RS(15)		RS(10)		RS(5)	
auto93.names	1.9640±	0.69	0.5045±	0.01 •	0.4029±	0.01 •	2.1210±	0.30	2.0398±	0.31	1.9364±	0.31
autoHorse.names	3.2094±	1.26	1.1297±	0.97 •	0.9626±	0.97 •	3.8606±	0.89	3.8277±	0.67	4.5211±	0.68 ○
autoMpg.names	13.2778±	5.00	17.9050±	2.61 ○	17.5662±	2.61 ○	17.2348±	2.61 ○	16.7957±	2.60	52.6709±	10.52 ○
autoPrice.names	2.6246±	1.02	0.8508±	0.71 •	0.7329±	0.71 •	2.4978±	0.71	2.5326±	0.53	3.1205±	0.75
baskball	1.5655±	0.86	1.5735±	0.78	1.4777±	0.78	1.6060±	0.69	1.8416±	0.60	1.9194±	0.64
bodyfat.names	3.6184±	1.27	2.2780±	1.83	2.0385±	1.83 •	3.8649±	1.44	4.0049±	1.13	5.6001±	0.64 ○
bolts	0.9012±	0.63	0.7395±	0.51	0.6571±	0.51	1.0105±	0.60	1.3959±	0.21 ○	1.4990±	0.14 ○
breastTumor	4.7626±	2.32	1.8017±	2.09 •	1.7195±	2.14 •	4.4978±	1.43	4.0813±	1.17	9.6909±	5.72 ○
cholesterol	5.6980±	3.45	3.9363±	4.09	3.9822±	4.08	5.8513±	3.80	6.0970±	4.46	14.8924±	7.96 ○
cleveland	4.6643±	2.58	1.5867±	1.60 •	1.3875±	1.60 •	4.7084±	0.84	4.4935±	0.84	11.2852±	3.62 ○
cloud	1.0600±	0.63	1.2142±	0.55	1.1205±	0.55	1.3661±	0.30	1.2787±	0.30	2.3111±	0.35 ○
cpu	1.7717±	1.11	0.8778±	0.66 •	0.7321±	0.66 •	0.6683±	0.85 •	3.8958±	0.51 ○	3.8508±	0.49 ○
detroit	1.4814±	0.55	0.5058±	0.33 •	0.4239±	0.33 •	1.3080±	0.52	1.3384±	0.44	1.4705±	0.45
echoMonths	1.6654±	0.63	0.9987±	0.76 •	0.8797±	0.76 •	1.5365±	0.33	1.2264±	0.01 •	2.2524±	0.32 ○
elusage	1.1918±	0.66	0.7621±	0.41	0.6843±	0.41	0.9985±	0.44	1.0568±	0.32	1.8957±	0.42 ○
fishcatch	2.1945±	0.89	0.6110±	0.34 •	0.5001±	0.34 •	0.4081±	0.37 •	2.8666±	0.57 ○	2.6901±	0.36
fruitfly	1.6387±	0.84	0.9206±	0.80	0.8055±	0.78 •	1.2279±	0.82	1.1586±	0.74	1.5212±	0.70
gascons	1.5059±	0.41	1.2224±	0.77	1.1370±	0.77	1.1237±	0.77	1.6855±	0.32	1.3918±	0.16
housing	13.0430±	9.38	10.1886±	8.87	9.4929±	8.87	9.0099±	8.87	27.4480±	9.64 ○	18.6362±	1.75
hungarian	6.2251±	1.22	8.1133±	1.68 ○	8.1711±	1.13 ○	8.1250±	0.40 ○	7.9078±	0.39 ○	19.4831±	4.82 ○
longley	1.2865±	0.55	0.8002±	0.39 •	0.7220±	0.39 •	1.0806±	0.35	1.0493±	0.29	1.4102±	0.28
lowbwt	1.5539±	1.04	1.1722±	0.89	1.0444±	0.89	2.0855±	0.41	1.9618±	0.41	7.7215±	1.72 ○
mbagrade	1.1909±	0.80	1.0788±	0.57	1.0129±	0.58	1.1002±	0.54	1.1949±	0.40	1.8837±	0.55 ○
meta	20.6206±	12.60	16.7876±	11.80	16.0775±	11.81	22.8163±	7.47	22.9367±	8.04	33.5039±	6.79 ○
pbc	11.4023±	3.18	6.9180±	6.28 •	6.5458±	6.28 •	13.2360±	2.46	12.7740±	2.44	44.4131±	20.96 ○
pharynx	3.1827±	1.85	1.2907±	0.01 •	1.0399±	0.01 •	6.1518±	1.94 ○	5.9801±	1.97 ○	7.1121±	0.22 ○
pollution	1.4285±	0.48	0.7544±	0.54 •	0.6718±	0.54 •	1.4066±	0.46	1.4931±	0.48	2.0283±	0.19 ○
pwLinear	3.2568±	0.87	4.0231±	0.72	3.8965±	0.72	3.8074±	0.64	3.7026±	0.63	3.9955±	0.66 ○
quake	1526.2992±	2007.88	2054.3738±	2040.03	1946.7816±	1899.94	2582.2793±	1878.03	2271.0889±	1737.78	2674.1716±	1169.13
schlvote	1.3568±	0.62	1.0765±	0.60	0.9952±	0.59	1.1920±	0.54	1.1597±	0.45	1.3362±	0.58
sensory	27.0773±	11.87	42.2451±	8.84 ○	41.5666±	8.84 ○	41.9842±	5.79 ○	41.2938±	5.80 ○	43.8216±	4.37 ○
servo	2.3701±	0.53	1.0345±	1.03 •	0.9232±	1.02 •	2.8666±	0.42	2.7543±	0.43	2.5921±	0.37
sleep	1.5765±	0.50	1.1308±	0.74	1.0457±	0.74 •	1.5479±	0.42	1.3495±	0.41	1.8250±	0.44
strike	34.2065±	14.39	28.4238±	16.65	27.6472±	16.66	34.2569±	10.93	34.4711±	15.88	45.6365±	7.05 ○
veteran	2.5879±	1.23	2.2981±	0.87	2.1968±	0.87	2.2016±	0.75	2.3615±	0.77	2.7658±	0.85
vineyard	1.1182±	0.69	0.5885±	0.47	0.5102±	0.48	0.6344±	0.61	1.4404±	0.36	1.5515±	0.49
Average	47.6272		61.7143		58.5431		77.5465		69.5551		84.4002	

○, • statistically significant improvement or degradation

4.2 Classification

In the last section, we described the application of the GPO algorithm for regression problems. In this section, the focus is on how to apply the GPO algorithm to classification problems. Although the class variable of classification learning is a categorical object rather than a numeric value, we can still use the Gaussian process regression approach for building the GP model to tune parameter values of several well-known algorithms. Generally, different tuning parameters will affect the performance of the final model. Some of the models are useful for prediction while some of them are not appropriate. It is impossible to enumerate through all the classification algorithms and their control parameters for finding the optimal model. Here, we will present support vector machines, decision tree learning, and additive logistic regression as representatives to test the effectiveness of the GPO algorithm.

True Class	Predicted Class	
	YES	NO
YES	TP	FN
NO	FP	TN

Figure 4.1: Confusion matrix table

Before running experiments, we should consider the evaluation measures which we will use later. There are many performance measures for classification evaluation in machine learning, such as accuracy, error rate, precision, recall [33] and the receiver operating characteristic curve (ROC) [8] [49]. Accuracy is a widely used method while assessing and comparing the performance of one learning algorithm to another, it is calculated as the percentage of the number of correct predictions over the total number of predictions [27]. The number of correct predictions is the count of the correct records predicted by the model and the total number of predictions includes the correct predictions and incorrect predictions. In the two-class classification case with classes "yes" and "no", these counts are shown as a confusion matrix table in Figure 4.2, where the true classes are presented in rows while the predicted classes are presented

in columns. Hence, there are four possible outcomes for a single prediction: true positive (TP), false negative (FN), false positive (FP) and true negative (TN). TP and TN are correct predictions where TP indicates the number of positive examples correctly predicted as yes and TN indicates the number of negative examples correctly predicted as no by the model. In contrast, FP refers to where the outcome is incorrectly predicted as yes when it is actually negative. Similarly, FN occurs when the outcome should be yes but is incorrectly predicted as negative. Both predictions are incorrect predictions. Consequently, the number of correct predictions is the sum of TP and TN, while the number of incorrect predictions is the sum of FN and FP.

The appropriate model to select not only depends on the accuracy of the classification prediction, but also the computational cost. The running time increases with the increase in the number of training examples and the number of attributes in an example. Hence, running time and search time are important measures to evaluate GPO for classifiers. In addition, good performance on a particular dataset cannot determine whether one optimization algorithm is better than another one. Thus, we are going to deploy several datasets with different sizes and compare the accuracy, runtime, and number of search runs among GPO, Grid Search and Random Search.

4.2.1 Classification Prediction Based on SMO

In Chapter 2, we briefly introduced the theoretical background and the general algorithm of SMO. In this section, we will focus on applying the GPO algorithm to optimise the parameters of SMO in order to obtain accurate classification predictions. First, we need to consider which two parameters we intend to optimize and these two parameters will be regarded as the two dimensions of the search space. Here, C_{base} of the SMO function is one parameter to be optimized. The *slack* parameter C in the classification permits individual samples to fall on the "wrong" side of the decision boundary. The Gaussian kernel or RBF function together with the SMO classifier is the most popular choice of kernel types by far. Another kernel function parameter we will use is γ_{base} in the RBF kernel. Thus the two parameters being optimized are C_{base} of the support vector classifier and γ_{base} of the RBF kernel respectively. Secondly, we need to

decide the parameter ranges before doing any search. In this case, we specify that these parameters range from 10^{-5} to 10^{5} respectively, where the exponent is increased with step size 1. Finally, as per the regression experiments, we fix the noise δ_{GPO} of the GP model at 0.01 and the γ_{GPO} value in the RBF kernel for the GP model at 10.

Table 4.10: The search configuration of GPO and Grid Search in classification using SMO

Common points in GPO and Grid Search	Base algorithm:SMO function Search parameters: γ of RBF kernel function in the classifier treated as x_1 and C of the base algorithm in the classifier treated as x_2 Search range: I$\in \{-5, -4, \ldots, 5\}$ Search step: 1 Search base: 10 Search expression: 10^I Search route: by columns
Difference in GPO and Grid Search	GPO algorithm builds GP model for predictive distribution, the parameters of which are the Gaussian noise δ and the width γ of the RBF kernel function. We specify the γ value equals 10 and the noise level δ as 0.01.

The SVM parameter selection algorithm was evaluated through the Experimenter interface of WEKA. Table 4.10 presents the experiment set-up for GPO and Grid Search in the SMO classification learning task. Both optimize classification accuracy. The datasets that are used for the experiments are all from the UCI machine learning repository.

Table 4.11 compares the accuracy of GPO and Grid Search algorithm on the datasets. The result of a single dataset is the average accuracy estimation over 100 results, because the system will run 10-fold cross-validation 10 times in all. The two sets of results show that GPO is comparable with Grid Search in terms of accuracy, because there is no statistical difference between these two results except in one case, the "vowel" dataset. The main reason for this special case is that the optimal point is located on the border of the grid when running 2-fold cross-validation, which avoid search going further. We also find that the result of the GPO algorithm is slightly better than Grid search for some datasets while in some situations the GPO algorithm is inferior to Grid search.

Table 4.11: Percent Correct of GPO and Grid Search in UCI Classification using SMO

Dataset	GPO	Grid Search
anneal	99.5106± 0.6946	99.4884± 0.7141
anneal.ORIG	89.3764± 2.9429	89.5880± 2.7189
arrhythmia	70.2449± 4.8209	70.2024± 5.3954
audiology	80.6719± 6.6037	80.8063± 6.6167
autos	76.7524± 8.6472	77.2762± 9.0314
balance-scale	99.7911± 1.1133	99.8559± 0.6068
breast-cancer	70.3325± 5.0278	71.6958± 6.3171
wisconsin-breast-cancer	96.5383± 2.0244	96.5385± 2.0944
horse-colic	83.7432± 5.3096	83.4707± 5.4324
horse-colic.ORIG	74.6704± 7.0865	75.1321± 6.7405
credit-rating	85.1449± 3.8887	85.5942± 3.9183
german-credit	75.4000± 3.5133	75.3200± 3.3208
pima-diabetes	76.7307± 4.4174	76.8209± 4.4732
ecoli	87.2923± 5.6084	87.2638± 5.8144
Glass	70.5671± 8.8021	70.5152± 9.4549
cleveland-14-heart-disease	82.4774± 5.7013	82.2839± 5.9605
hungarian-14-heart-disease	82.4000± 6.5175	82.5989± 5.8760
heart-statlog	83.2593± 6.2394	83.2222± 6.0814
hepatitis	83.6625± 8.5069	83.6958± 8.5354
hypothyroid	97.6858± 0.7089	97.6672± 0.7210
ionosphere	93.9063± 4.0718	94.0206± 3.8949
iris	95.8000± 4.7995	95.4000± 5.1635
kr-vs-kp	99.6560± 0.3280	99.6622± 0.3374
labor	90.2667±12.5232	90.1667±12.3581
lymphography	83.7190± 8.3062	83.1095± 8.6807
mushroom	100.0000± 0.0000	100.0000± 0.0000
optdigits	99.2544± 0.3574	99.2420± 0.3555
pendigits	99.6388± 0.1902	99.6388± 0.1902
primary-tumor	44.3699± 6.2497	44.8431± 6.3497
segment	97.0779± 1.1921	97.0866± 1.1960
sick	96.8929± 0.7723	96.9008± 0.7147
sonar	86.1143± 7.8486	86.3048± 7.6206
soybean	93.4706± 2.4040	93.3796± 2.5454
vehicle	84.7842± 3.0714	84.8190± 3.1110
vote	96.1094± 2.7120	95.9942± 2.7101
vowel	92.4040± 3.0708	99.2828± 0.9977 ∘
waveform	86.3460± 1.5114	86.3220± 1.4583
zoo	95.9545± 5.6180	96.0545± 5.5999

∘, • statistically significant improvement or degradation

However, it is encouraging that the two approaches have comparable accuracy using a paired t-test.

Although accuracy plays a substantial role in classification predictions, one cannot decide which algorithm is better based solely on this measure. Tables 4.12 and 4.13 give the number of runs of 2-fold cross-validation and 10-fold cross-

Table 4.12: Number of 2-Fold Cross-validation Runs in GPO and Grid Search in UCI classification

Dataset	GPO	Grid Search
anneal	12.6000±3.3333	121.0 ± 0.0 ∘
anneal.ORIG	16.0800±5.0526	121.0 ± 0.0 ∘
arrhythmia	12.1400±3.7686	121.0 ± 0.0 ∘
audiology	12.8700±2.8343	121.0 ± 0.0 ∘
autos	14.2500±3.0957	121.0 ± 0.0 ∘
balance-scale	18.1700±3.8219	121.0 ± 0.0 ∘
breast-cancer	26.6200±9.4386	121.0 ± 0.0 ∘
wisconsin-breast-cancer	14.0900±5.2167	121.0 ± 0.0 ∘
horse-colic	12.7400±3.7026	121.0 ± 0.0 ∘
horse-colic.ORIG	13.5600±3.8490	121.0 ± 0.0 ∘
credit-rating	15.6500±4.5845	121.0 ± 0.0 ∘
german-credit	12.9000±4.1084	121.0 ± 0.0 ∘
pima-diabetes	14.5400±4.0487	121.0 ± 0.0 ∘
ecoli	13.9000±4.2319	121.0 ± 0.0 ∘
Glass	18.1800±3.9705	121.0 ± 0.0 ∘
cleveland-14-heart-disease	13.7600±4.4338	121.0 ± 0.0 ∘
hungarian-14-heart-disease	12.1800±3.7615	121.0 ± 0.0 ∘
heart-statlog	11.3100±2.3342	121.0 ± 0.0 ∘
hepatitis	13.7400±6.5298	121.0 ± 0.0 ∘
hypothyroid	12.2400±1.0456	121.0 ± 0.0 ∘
ionosphere	18.7300±4.4378	121.0 ± 0.0 ∘
iris	11.9100±2.6670	121.0 ± 0.0 ∘
kr-vs-kp	17.6100±2.3177	121.0 ± 0.0 ∘
labor	10.9800±1.7407	121.0 ± 0.0 ∘
lymphography	12.0400±2.9845	121.0 ± 0.0 ∘
mushroom	10.5700±1.6407	121.0 ± 0.0 ∘
optdigits	21.9100±3.1849	121.0 ± 0.0 ∘
pendigits	20.3500±1.4590	121.0 ± 0.0 ∘
primary-tumor	11.4200±2.3621	121.0 ± 0.0 ∘
segment	16.7000±2.9729	121.0 ± 0.0 ∘
sick	18.3200±5.0869	121.0 ± 0.0 ∘
sonar	16.1700±3.0717	121.0 ± 0.0 ∘
soybean	12.4300±3.1631	121.0 ± 0.0 ∘
vehicle	17.5500±3.3826	121.0 ± 0.0 ∘
vote	12.5600±3.4650	121.0 ± 0.0 ∘
vowel	11.9000±0.3015	121.0 ± 0.0 ∘
waveform	15.4700±4.8813	121.0 ± 0.0 ∘
zoo	10.7800±2.2046	121.0 ± 0.0 ∘

∘, • statistically significant improvement or degradation

validation respectively. GPO merely performs several 2-fold cross-validations while Grid Search has to perform 2-fold cross-validations for every point in the parameter space. It can be seen in Table 4.12 that the GPO algorithm utilizes no more than 27 samples to find the C and γ values that gave the best classification accuracy during the initial search process. By contrast, Grid Search has to run

Table 4.13: Number of 10-Fold Cross-validation Runs in GPO and Grid Search in UCI classification using SMO

Dataset	GPO	Grid Search
anneal	8.4200±7.1706	4.0800±7.0920
anneal.ORIG	8.1400±6.0420	10.9400±4.4672
arrhythmia	5.4500±7.4473	7.6300±8.3165
audiology	13.5600±5.1253	0.7000±3.2208 •
autos	10.2700±6.4289	4.4800±7.2230
balance-scale	5.1400±5.1287	0.4500±1.9714 •
breast-cancer	7.2800±5.1602	9.4400±5.4482
wisconsin-breast-cancer	13.0900±5.3788	8.7200±7.0783
horse-colic	5.3300±6.7466	12.8500±6.6718 ○
horse-colic.ORIG	8.5000±7.4894	6.0900±8.8775
credit-rating	7.6300±7.9387	12.4600±5.6985
german-credit	4.3200±6.3084	9.6200±5.4435 ○
pima-diabetes	8.9700±6.7890	7.7300±6.1936
ecoli	12.5600±6.5649	9.4900±7.4880
Glass	9.9700±5.3588	9.4700±5.7427
cleveland-14-heart-disease	7.3200±7.3442	7.0200±7.7577
hungarian-14-heart-disease	5.9700±6.9347	6.3500±6.7545
heart-statlog	7.3600±7.6204	6.5500±8.7552
hepatitis	5.4600±8.4536	6.4300±8.7770
hypothyroid	0.1800±1.2663	0.3600±1.7725
ionosphere	12.0700±4.0856	10.4000±4.8990
iris	15.4000±8.5552	4.3900±8.6257 •
kr-vs-kp	9.8500±6.8922	7.6900±8.1992
labor	8.7900±7.5829	4.3500±7.7282
lymphography	7.5700±8.0092	3.0400±6.7028
mushroom	13.7000±2.1391	0.0000±0.0000 •
optdigits	10.1100±4.1485	5.7600±5.9342 •
pendigits	9.6800±1.9688	9.5900±2.1932
primary-tumor	1.6900±4.4488	10.4000±7.0825 ○
segment	9.7800±5.0982	9.7200±5.1953
sick	7.0300±6.5342	2.7500±5.0219
sonar	12.5100±6.3683	4.3500±8.1728 •
soybean	5.3200±6.8886	6.7200±7.0526
vehicle	9.8400±4.1065	9.7900±4.2529
vote	10.3700±7.2748	5.9200±7.3782
vowel	0.0000±0.0000	6.8600±7.1039 ○
waveform	8.6000±7.6317	10.6900±6.5516
zoo	17.4800±3.0067	0.1400±1.4000 •

○, • statistically significant improvement or degradation

121 iterations to obtain the best accuracy in the initial searches, because there are 11 values for each parameter in the range and thus the system has 11×11 samples in the search space.

Once the "best" point is decided after the 2-fold cross-validation runs, the system will run 10-fold cross-validation around the point which is the center of

the new range. In the five of the thirty-eight datasets in Table 4.13 marked by solid black circles, Grid search is statistical significantly better than the GPO algorithm in terms of 10-fold cross-validation, whereas for four datasets marked by hollow circles the GPO algorithm performs significantly better than Grid Search. The rest of the datasets show comparable results for GPO and Grid Search. This former situation occurs when the so-called "best" point identified using the GP model is not actually close to the best one, and the system has to jump to other sub-grids to acquire the optimal point through 10-fold cross-validation. Conversely, the latter situation implies that the system has already found a point close to the optimal point after the 2-fold cross-validation runs. Comparing the 2-fold and 10-fold cross-validation runs of the GPO and Grid Search algorithms, GPO often converges to the optimal area more quickly than the Grid Search algorithm.

The training time for the predictive classification problems on each dataset involves evaluating data, performing cross-validations and building the final predictive model. It is shown in Table 4.14. The training time obviously varies for different datasets. There are several reasons that affect the training time. On the one hand, it depends on the nature of the dataset, such as the number of instances and the number of attributes of each instance. On the other hand, it also depends on the complexity of the base algorithm to be used: the easier the algorithm, the quicker the experiment. Beyond that, we should consider the number of runs performed, which includes 2-fold cross-validation as well as 10-fold cross-validation. It is clear that the time consumed by a 10-fold cross-validation is much greater than that of a 2-fold cross-validation. As shown in Table 4.14, the training time spent on the GPO algorithm is statistically less than that spent on the Grid Search algorithm on most of the datasets. For instance, the "horse-colic" dataset requires plenty of training time for Grid Search in that it initially performs 121 2-fold cross-validation runs plus 13 10-fold cross-validation runs afterwards. Hence, the GPO algorithm performs better than Grid Search in terms of time, which can reduce the time cost.

Through the above analysis between GPO and Grid Search, it is clear to see that Grid Search has obvious drawbacks in terms of runtime and number of cross-validation runs, even if it has comparable accuracy to GPO. In order

Table 4.14: Training Time of GPO and Grid Search in UCI classification using SMO

Dataset	GPO		Grid Search		
anneal	50.4535±	37.8400	110.2238±	34.6561	○
anneal.ORIG	178.3583±	168.3954	351.6973±	199.6062	○
arrhythmia	87.9929±	75.4111	344.9478±	79.9634	○
audiology	318.4004±	99.3345	399.7094±	67.4793	
autos	21.4786±	10.4381	39.7159±	11.1331	○
balance-scale	66.0901±	56.8101	28.7033±	23.3578	
breast-cancer	6.3967±	12.6444	17.8793±	20.8893	
wisconsin-breast-cancer	13.5801±	9.1229	18.6931±	8.2391	
horse-colic	7.0305±	8.3607	18.0689±	8.2625	○
horse-colic.ORIG	17.9277±	12.4802	33.6782±	13.3403	○
credit-rating	74.6568±	141.6875	80.4682±	75.3732	
german-credit	368.1510±	699.0174	390.9259±	585.3184	
pima-diabetes	348.2567±	462.2978	528.3237±	484.0569	
ecoli	39.1734±	17.6264	77.2833±	18.5122	○
Glass	36.8025±	22.5593	60.7217±	23.3581	○
cleveland-14-heart-disease	12.7705±	21.4263	20.4910±	24.9281	
hungarian-14-heart-disease	21.5160±	36.2389	29.2801±	31.8227	
heart-statlog	12.9722±	21.4542	15.7295±	18.6052	
hepatitis	2.1328±	2.7104	4.3191±	2.6056	
hypothyroid	254.5471±	536.4674	930.2017±	661.6887	○
ionosphere	7.0421±	3.2411	10.1883±	2.5223	○
iris	4.8671±	2.1852	7.2241±	2.2580	○
kr-vs-kp	906.2570±	593.8338	1256.2260±	699.7527	
labor	1.1307±	0.6704	2.3982±	0.6854	○
lymphography	6.0971±	4.7855	13.2572±	3.9421	○
mushroom	10062.5831±	4015.9588	20445.5426±	376.7559	○
optdigits	1712.6954±	462.3567	7366.6403±	633.4851	○
pendigits	3398.5689±	498.5419	16873.0792±	574.4115	○
primary-tumor	71.2509±	77.6824	495.6887±	116.4245	○
segment	241.0457±	130.1995	646.4394±	133.5034	○
sick	3011.5108±	2826.1437	1650.2531±	2148.5649	
sonar	4.7662±	1.9875	5.1357±	2.3372	
soybean	156.3681±	124.9832	611.8247±	122.9058	○
vehicle	745.5264±	422.8238	798.9005±	428.6658	
vote	5.9170±	4.1354	8.6115±	3.8269	
vowel	32.3381±	0.8078	326.4173±	93.5856	○
waveform	2681.5446±	2266.6288	7000.6817±	2261.8583	○
zoo	34.2251±	4.9468	40.1095±	3.1077	○

○, ● statistically significant improvement or degradation

to thoroughly understand the behaviour of GPO, we compare it with Random Search using the same measures: accuracy, training time and number of cross-validation runs. It is self-evident that Random Search should be cheaper than Grid Search in exploring the search space.

There are more than 100 points in the grid in this experiment. GPO only

Table 4.15: Number of 2-Fold Cross-validation Runs in GPO and Random Search in UCI classification using SMO

Dataset	GPO	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	11.04±1.88	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0	5±0.0 •
zoo	10.56±1.85	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0	5±0.0 •
lymphography	11.96±2.99	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0	5±0.0 •
iris	11.82±2.47	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0 •	5±0.0 •
hepatitis	14.60±7.51	25±0.0 ∘	20±0.0	15±0.0	10±0.0	5±0.0 •
autos	14.00±3.25	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0 •	5±0.0 •
sonar	16.26±3.18	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0 •	5±0.0 •
Glass	18.24±4.09	25±0.0 ∘	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
audiology	12.86±2.86	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0 •	5±0.0 •
heart-statlog	11.48±2.75	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0	5±0.0 •
breast-cancer	26.14±9.27	25±0.0	20±0.0	15±0.0 •	10.0±0.0 •	5±0.0 •
hungarian	12.44±3.94	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0	5±0.0 •
cleveland	14.76±4.81	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0 •	5±0.0 •
ecoli	14.24±4.01	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0 •	5±0.0 •
primary-tumor	11.36±2.02	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0	5±0.0 •
ionosphere	18.80±4.66	25±0.0 ∘	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
horse-colic.ORIG	13.80±4.05	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0 •	5±0.0 •
horse-colic	12.74±3.90	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0	5±0.0 •
vote	12.64±3.35	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0 •	5±0.0 •
arrhythmia	12.62±4.03	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0	5±0.0 •
balance-scale	18.26±3.66	25±0.0 ∘	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
soybean	12.26±3.24	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0	5±0.0 •
credit-rating	15.78±4.51	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0 •	5±0.0 •
wisconsin	14.06±5.17	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0 •	5±0.0 •
pima-diabetes	14.26±3.82	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0 •	5±0.0 •
vehicle	18.10±3.16	25±0.0 ∘	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •
anneal	12.22±3.17	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0	5±0.0 •
anneal.ORIG	16.52±5.17	25±0.0 ∘	20±0.0	15±0.0	10±0.0 •	5±0.0 •
vowel	11.90±0.30	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0 •	5±0.0 •
german-credit	12.64±3.65	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0	5±0.0 •
segment	16.58±3.12	25±0.0 ∘	20±0.0 ∘	15±0.0	10±0.0 •	5±0.0 •
kr-vs-kp	17.54±2.02	25±0.0 ∘	20±0.0 ∘	15±0.0 •	10±0.0 •	5±0.0 •
sick	17.56±4.87	25±0.0 ∘	20±0.0	15±0.0	10±0.0 •	5±0.0 •
hypothyroid	12.00±0.20	25±0.0 ∘	20±0.0 ∘	15±0.0 ∘	10±0.0 •	5±0.0 •
optdigits	22.00±3.04	25±0.0 ∘	20±0.0	15±0.0 •	10±0.0 •	5±0.0 •

∘, • statistically significant improvement or degradation

needs to evaluate approximately 15 points to obtain optimal performance, while Grid Search has to evaluate all the data points of the grid, see Table 4.12. In Random Search, the number of data points to be evaluated depends on the aim of the user. Tables 4.15 and 4.16 list the results of Random Search when the number of grid locations is specified as 25, 20, 15, 10 and 5 respectively. With the decreasing number of grid locations, the data points to be explored initially using 2-fold cross-validation are reduced as well.

Table 4.16: Number of 10-Fold Cross-validation Runs in GPO and Random Search in UCI classification using SMO

Dataset	GPO	Grid Search	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	9.08±7.17	3.34±6.62	12.08± 7.10	7.68±8.29	14.34±3.50	14.34±3.50	14.60±4.44
zoo	17.74±2.10	0.00±0.00 •	16.44± 5.60	0.28±1.98 •	10.10±2.91 •	10.10±2.91 •	9.64±2.35 •
lymphography	8.78±8.61	1.92±5.12	13.02± 7.76	10.82±7.49	13.98±3.71	13.98±3.71	15.60±4.69
iris	16.10±8.56	3.38±7.92 •	9.60±11.06	4.10±8.81 •	13.46±6.99	10.42±4.73	10.42±4.73
hepatitis	6.14±8.58	6.96±9.14	16.28± 5.90 ○	12.68±5.16	13.78±3.56 ○	14.90±3.22 ○	17.04±5.04 ○
autos	9.66±6.93	3.28±6.72 •	9.14± 8.73	1.72±4.18 •	10.48±3.38	10.48±3.38	10.48±3.38
sonar	12.80±6.16	6.04±9.17	14.74± 5.63	11.84±5.79	12.38±5.33	17.70±5.69	17.80±5.62
Glass	10.40±4.49	9.42±5.04	10.16± 3.38	10.16±3.38	10.70±2.21	9.18±0.72	9.18±0.72
audiology	13.40±5.34	0.48±2.45 •	7.20± 8.84	1.80±4.46 •	9.42±1.83	9.42±1.83	9.46±2.02
heart-statlog	8.22±8.23	5.64±8.89	13.68± 7.02	12.56±7.44	14.84±4.43 ○	14.18±3.41	15.90±4.81 ○
breast-cancer	6.82±5.32	9.86±5.23	11.14± 4.78	12.78±3.59 ○	12.18±4.79	11.94±4.46 ○	12.08±2.33 ○
hungarian	5.62±7.13	5.30±6.84	8.32± 8.13	6.50±7.23	11.92±3.20 ○	12.00±3.30 ○	12.00±3.75 ○
cleveland	8.46±7.13	7.84±7.72	10.52± 6.45	9.52±6.37	13.18±2.83	13.18±2.83	13.46±3.35
ecoli	13.46±6.43	10.06±7.01	4.04± 7.27 •	2.50±6.48 •	11.82±4.55	10.66±1.76	10.66±1.76
primary-tumor	1.30±3.62	9.80±6.98 ○	14.02± 4.71 ○	13.44±4.92 ○	14.20±3.65 ○	14.20±3.65 ○	14.18±4.24 ○
ionosphere	11.82±3.85	9.76±4.83	12.30± 3.81	12.40±3.79	12.40±3.79	19.48±5.02 ○	19.36±4.88 ○
horse-colic.ORIG	8.56±7.83	5.84±8.53	14.60± 6.03	12.60±6.94	14.78±3.95 ○	14.78±3.95 ○	15.16±4.56 ○
horse-colic	5.22±6.70	13.08±6.68 ○	14.10± 3.39 ○	14.04±3.37 ○	14.04±3.37 ○	14.04±3.37 ○	14.18±3.77 ○
vote	9.88±7.55	5.64±7.18	13.78± 4.46	12.98±5.04	14.34±2.62	14.34±2.62	14.28±3.98
arrhythmia	5.38±7.03	7.28±7.50	14.88± 6.01 ○	13.70±5.13 ○	14.90±3.17 ○	14.90±3.17 ○	15.60±4.12 ○
balance-scale	4.60±5.09	0.36±1.78 •	9.26± 1.84 ○	9.28±1.98 ○	9.28±1.98 ○	9.48±2.41 ○	9.28±1.98 ○
soybean	5.42±6.56	6.04±6.73	13.28± 3.43 ○	13.70±3.94 ○	14.36±2.83 ○	14.26±2.75 ○	13.70±2.90 ○
credit-rating	8.16±8.17	12.70±5.75	12.42± 3.65	13.62±2.84	13.90±2.05	14.04±2.29	14.24±3.24 ○
wisconsin	13.44±4.97	8.40±7.01	12.32± 5.49	10.76±6.05	12.36±4.24	14.28±2.78	15.08±3.93
pima-diabetes	9.00±6.45	7.98±6.36	10.48± 6.42	7.72±6.20	10.86±3.36	10.34±1.98	10.34±1.98
vehicle	10.06±4.02	9.92±3.61	12.00± 1.56	12.00±1.56	12.00±1.56	10.44±2.82	10.68±2.94
anneal	8.20±7.29	4.02±7.01	1.92± 5.61 •	0.64±2.62 •	9.40±1.37	9.40±1.37	9.40±1.37
anneal.ORIG	8.06±6.10	10.48±4.67	9.00± 6.55	3.60±6.44	12.94±3.72	12.62±3.52	12.62±3.52
vowel	0.00±0.00	6.36±7.02 ○	9.10± 0.71 ○	9.10±0.71 ○	9.10±0.71 ○	9.18±0.72 ○	9.18±0.72 ○
german-credit	4.18±6.21	9.68±4.83 ○	9.60± 5.85	9.32±5.97	12.70±2.86 ○	12.70±2.86 ○	12.18±2.31 ○
segment	9.64±4.78	9.22±5.13	9.88± 1.57	9.88±1.57	9.88±1.57	9.06±0.42	9.06±0.42
kr-vs-kp	9.60±7.26	7.80±8.43	16.94± 2.61 ○	12.46±4.35	12.46±4.35	12.46±4.35	12.46±4.35
sick	7.82±6.48	2.34±4.81	5.76± 5.39	5.76±5.39	9.88±1.99	9.50±1.81	9.50±1.81
hypothyroid	0.00±0.00	0.00±0.00	10.38± 1.51 ○	10.38±1.51 ○	10.38±1.51 ○	9.00±0.00 ○	9.00±0.00 ○
optdigits	9.72±4.05	5.76±5.38	10.26± 3.15	10.54±2.98	10.54±2.98	13.00±2.91 ○	13.00±2.91 ○

○, • statistically significant improvement or degradation

Tables 4.15 and 4.16 show the number of 2-fold cross-validation runs and 10-fold cross-validation runs respectively. The main difference in Table 4.15 is that the number of cross-validation runs in GPO is determined by the stopping criterion rather than the user, that is, we can not know how many cross-validations will be used during the process. But fortunately, the GPO algorithm does not waste time on unnecessary data points and only explores a small part of the space to output the final result. Note that the general search process is completely the same when running 10-fold cross-validation both for GPO and for Random Search. Once a near-optimal data point has been found after 2-fold cross-validation, they will search neighbouring points around the near-optimal data and determine whether any improvement occurs.

Let us focus on two columns in Table 4.16, which are GPO and Random Search with 15 grid locations, because the average number of data points searched in GPO is mostly close to 15. There are 12 out of 35 datasets marked by "circles" that have statistically significant improvement in 10-fold cross-validation runs in GPO. It implies that in some cases, the near-optimal point found by GPO using 2-fold cross-validation is better than that found by Random Search under the same number of points searched. The reason is that Random Search can not guarantee that the near-optimal point in each run is truly a good point, whereas the near-optimal point found through GPO is likely to be a high-quality one in each run of the process.

Table 4.17 shows the accuracy using GPO and Random Search. It is encouraging that GPO does not fall behind in terms of accuracy in support vector machine parameter optimization in most cases; however, it does not outperform the other algorithms here. Grid Search is a full-scale method, which can consider every point in the search space, but the good performance of Random Search is surprising.

Table 4.18 presents an interesting phenomenon involving training time for larger and smaller datasets when comparing GPO and Random Search. Note that the rows are sorted by the number of training instances. We find that solid dots generally appear in the upper part of this table while circles often appear in the lower part. For smaller datasets, GPO seems to have no improvement in training time, and generally consumes more time than Random Search.

Table 4.17: Percent Correct of GPO and Random Search in UCI Classification using SMO

Dataset	GPO	Grid Search	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	88.13±14.45	88.13±14.45	87.60±14.82	86.60±15.10	86.47±13.97	86.47±13.97	87.87±14.38
zoo	95.87± 5.97	96.07± 5.93	95.87± 5.97	96.07± 5.93	95.87± 5.97	95.87± 5.97	95.87± 5.97
lymphography	83.77± 9.04	82.96± 9.60	83.77± 8.75	83.90± 8.64	84.03± 8.62	84.03± 8.62	84.17± 8.91
iris	95.60± 4.97	95.20± 5.56	95.87± 4.64	95.87± 4.84	95.87± 4.84	96.00± 4.86	96.00± 4.86
hepatitis	83.28± 8.36	83.63± 8.51	83.78± 8.51	83.57± 8.15	83.81± 7.57	84.96± 7.64	83.79± 8.11
autos	76.88± 8.70	77.16± 9.53	76.31± 9.92	75.61± 9.51	76.10± 9.02	76.10± 9.02	76.10± 9.02
sonar	85.57± 7.85	85.49± 6.76	85.76± 6.97	85.38± 7.48	86.14± 7.22	84.50± 7.81	83.84± 7.88
Glass	70.56± 8.82	70.10± 9.61	69.90± 9.40	69.90± 9.40	70.38± 9.24	69.16± 9.16	69.16± 9.16
audiology	80.74± 6.55	81.10± 6.55	80.75± 6.70	81.37± 7.20	81.37± 7.21	81.37± 7.21	81.37± 7.21
heart-statlog	83.33± 5.03	82.89± 4.96	82.52± 5.89	83.04± 4.85	82.89± 4.84	82.96± 4.96	82.44± 5.93
breast-cancer	70.79± 5.88	71.75± 6.56	71.54± 7.12	72.17± 7.07	72.17± 6.62	72.17± 6.64	72.10± 6.70
hungarian	82.54± 6.70	82.95± 6.30	83.16± 6.15	83.30± 6.48	83.09± 6.49	83.09± 6.49	83.49± 6.38
cleveland	82.38± 5.94	82.05± 6.19	82.11± 6.20	82.18± 6.33	82.97± 6.22	82.97± 6.22	82.44± 6.02
ecoli	87.21± 6.10	87.27± 6.24	87.50± 5.92	87.44± 5.99	87.63± 5.79	87.51± 5.82	87.51± 5.82
primary-tumor	44.19± 5.76	44.54± 5.62	44.72± 6.04	44.60± 5.97	44.84± 5.98	44.84± 5.98	44.95± 6.18
ionosphere	94.48± 4.08	94.42± 4.07	94.37± 4.08	94.37± 4.08	94.37± 4.08	93.74± 4.28	93.74± 4.28
horse-colic.ORIG	74.95± 6.32	75.38± 5.88	75.49± 6.23	74.73± 6.27	75.11± 6.45	75.11± 6.45	75.60± 6.19
horse-colic	83.52± 5.84	83.46± 5.81	83.52± 5.62	83.41± 5.45	83.41± 5.45	83.41± 5.45	83.46± 5.81
vote	96.08± 2.76	95.99± 2.74	96.32± 2.98	96.13± 3.06	96.23± 2.97	96.23± 2.97	96.60± 2.75
arrhythmia	70.00± 5.54	69.87± 6.08	70.18± 6.09	70.18± 6.09	70.45± 6.10	70.45± 6.10	70.40± 6.10
balance-scale	99.68± 1.49	99.87± 0.63	99.87± 0.63	99.87± 0.63	99.87± 0.63	99.07± 1.34	99.04± 1.34
soybean	93.35± 2.43	93.38± 2.56	93.82± 2.47	93.56± 2.57	93.61± 2.55	93.61± 2.55	93.73± 2.50
credit-rating	85.25± 3.64	85.71± 3.65	85.04± 3.72	84.87± 3.73	84.93± 3.69	84.87± 3.75	84.99± 3.58
wisconsin	96.48± 1.98	96.54± 2.02	96.48± 2.09	96.48± 2.09	96.51± 2.05	96.51± 2.00	96.45± 2.01
pima-diabetes	76.98± 4.59	77.11± 4.44	77.13± 4.29	77.08± 4.41	76.87± 4.64	76.77± 4.77	76.77± 4.77
vehicle	85.30± 3.11	84.94± 3.16	85.32± 3.09	85.32± 3.09	85.32± 3.09	83.71± 3.66	83.64± 3.62
anneal	99.44± 0.75	99.40± 0.75	99.47± 0.72	99.49± 0.72	99.36± 0.81	99.36± 0.81	99.36± 0.81
anneal.ORIG	89.71± 3.09	89.76± 2.75	89.40± 3.15	89.22± 2.90	89.78± 2.77	89.87± 2.85	89.87± 2.85
vowel	92.46± 2.87	99.35± 0.95 ○	99.45± 0.94 ○	99.45± 0.94 ○	99.45± 0.94 ○	97.31± 1.77 ○	97.31± 1.77 ○
german-credit	75.52± 3.39	75.66± 3.09	75.50± 3.44	75.44± 3.37	75.50± 3.44	75.50± 3.44	75.46± 3.47
segment	97.20± 1.09	97.22± 1.09	97.12± 1.18	97.12± 1.18	97.12± 1.18	96.98± 0.84	96.98± 0.84
kr-vs-kp	99.66± 0.35	99.65± 0.36	99.65± 0.36	99.57± 0.40	99.57± 0.40	99.57± 0.40	99.57± 0.40
sick	96.94± 0.84	96.88± 0.82	96.88± 0.77	96.88± 0.77	96.93± 0.82	96.89± 0.81	96.89± 0.81
hypothyroid	97.66± 0.67	97.66± 0.67	97.57± 0.74	97.57± 0.74	97.57± 0.74	97.67± 0.67	97.67± 0.67
optdigits	99.24± 0.37	99.23± 0.37	99.25± 0.38	99.21± 0.38	99.21± 0.38	99.22± 0.42	99.22± 0.42

○, ● statistically significant improvement or degradation

Table 4.18: Training Time of GPO and Random Search in UCI classification using SMO

Dataset	GPO	Grid Search	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	0.39± 0.24	0.70± 0.19 ○	0.50± 0.25	0.31± 0.22	0.47± 0.09	0.44± 0.09	0.41± 0.12
zoo	9.28± 0.91	11.93± 2.36 ○	9.56± 2.54	1.93± 0.85 ●	5.81± 1.32 ●	5.36± 1.29 ●	4.67± 1.04 ●
lymphography	2.20± 1.63	3.72± 1.02 ○	3.43± 1.71	2.54± 1.35	3.07± 0.67	2.93± 0.67	2.91± 0.75
iris	1.55± 0.74	2.22± 0.94	1.31± 1.20	0.61± 0.70 ●	1.19± 0.58	0.85± 0.38 ●	0.78± 0.37 ●
hepatitis	1.09± 1.33	1.94± 1.35	2.61± 1.58 ○	1.94± 1.25	2.13± 1.17	2.48± 1.13 ○	2.06± 1.03
autos	6.74± 3.76	13.39± 5.22 ○	8.06± 5.67	2.77± 2.42 ●	7.63± 1.73	7.26± 1.73	6.34± 1.64
sonar	2.63± 1.13	2.75± 1.42	3.05± 1.29	2.13± 0.92	2.27± 1.13	4.10± 1.10 ○	3.17± 0.81
Glass	14.51± 10.18	23.79± 13.09	31.45± 11.92 ○	26.81± 8.64 ○	27.34± 7.30 ○	6.10± 2.51 ●	5.36± 2.16 ●
audiology	82.01± 26.83	114.70± 27.17 ○	60.04± 47.20	26.23± 21.94 ●	60.04± 9.49 ●	56.30± 9.55 ●	51.67± 10.40 ●
heart-statlog	8.57± 13.15	6.77± 8.03	10.49± 15.83	8.45± 13.74	14.95± 17.02	17.00± 18.14	13.15± 13.86
breast-cancer	3.24± 7.99	10.33± 13.28	7.40± 11.72	6.92± 10.53 ○	6.46± 10.59	6.78± 10.76	4.83± 11.14
hungarian-14-heart-disease	9.68± 15.62	11.19± 11.19	16.74± 21.81	13.32± 18.68	29.67± 14.86 ○	31.60± 14.45 ○	21.54± 12.38
cleveland-14-heart-disease	6.63± 10.47	9.22± 10.14	10.82± 15.03	9.72± 14.68	16.83± 17.57	16.81± 17.67	10.79± 13.59
ecoli	12.12± 5.95	24.94± 7.40 ○	6.16± 4.72 ●	4.20± 4.27 ●	9.98± 3.05	8.50± 1.66	7.73± 1.49
primary-tumor	17.28± 16.48	131.00± 31.99 ○	84.92± 27.07 ○	72.82± 23.48 ○	72.56± 18.65 ○	68.87± 18.65 ○	63.50± 20.42 ○
ionosphere	3.93± 2.26	5.57± 1.87 ○	4.19± 1.74	3.40± 1.18	3.27± 1.18	9.16± 4.97 ○	7.53± 4.00
horse-colic.ORIG	10.41± 7.45	22.92± 11.15 ○	19.63± 9.15 ○	14.65± 6.61	16.42± 3.54 ○	15.93± 3.60	14.16± 3.68
horse-colic	3.39± 3.91	9.94± 4.97 ○	9.54± 6.04 ○	8.10± 5.30	7.68± 5.24	7.59± 5.26	5.69± 4.34
vote	2.61± 1.92	4.30± 2.17 ○	4.44± 2.63 ○	3.54± 1.74	3.74± 1.46	3.49± 1.38	2.97± 1.39
arrhythmia	36.96± 28.99	158.90± 30.98 ○	91.07± 36.09 ○	70.25± 19.34 ○	70.25± 12.38 ○	64.63± 12.36 ○	63.29± 14.93 ○
balance-scale	27.97± 26.54	13.96± 9.30	31.06± 9.13	26.47± 7.18	26.01± 7.15	55.22± 9.19 ○	44.62± 5.79
soybean	57.89± 43.42	238.74± 41.40 ○	134.97± 44.55 ○	111.80± 25.05 ○	108.13± 19.42 ○	98.47± 18.86 ○	91.58± 19.07 ○
credit-rating	42.93± 74.53	49.98± 48.68	138.23± 241.26	144.03± 147.14 ○	145.22± 145.83 ○	149.12±144.20 ○	89.36±122.44
wisconsin-breast-cancer	6.34± 4.45	9.54± 4.90	5.55± 3.23	4.28± 2.38	5.79± 3.67	10.40± 6.86	7.15± 3.76
pima-diabetes	188.15± 254.34	281.46± 277.18	117.53± 233.62	79.40± 160.39	105.22± 179.08	213.39±300.16	168.12±227.88
vehicle	386.92± 211.64	532.52± 243.51 ○	700.40± 261.99 ○	562.75± 124.09	561.38± 124.20	189.08±203.32	180.32±184.73
anneal	24.14± 18.44	60.62± 16.32 ○	14.79± 14.04	8.87± 4.62 ●	22.13± 2.41	19.45± 2.54	17.12± 2.18
anneal.ORIG	82.35± 80.00	189.93± 129.40 ○	101.03± 122.33	73.04± 125.11	233.16± 118.53 ○	221.04±114.93 ○	180.48± 88.49 ○
vowel	15.12± 0.40	161.31± 53.57 ○	98.45± 24.57 ○	79.99± 5.94 ○	75.60± 5.92 ○	58.93± 5.77 ○	51.43± 3.23 ○
german-credit	127.08± 228.14	133.86± 154.65	135.72± 238.49	122.75± 240.37	192.01± 288.72	195.39±297.16	172.97±244.14
segment	125.34± 72.20	351.04± 76.79 ○	279.15± 82.20 ○	221.68± 26.31 ○	211.06± 26.41 ○	57.15± 11.05 ●	44.76± 8.84 ●
kr-vs-kp	531.57± 428.03	807.22± 500.16 ○	1050.32± 372.25 ○	523.57± 190.82	507.79± 190.99	490.86±190.91	420.69±184.87
sick	1885.81±1633.23	814.41±1063.36	1010.77±1620.65	862.90±1318.46	1092.70±1205.87	621.28±512.12 ●	528.01±384.57 ●
hypothyroid	100.05± 34.95	488.62± 118.80 ○	2835.77±1050.42 ○	2243.68± 351.45 ○	2231.26± 351.64 ○	540.95± 63.14 ○	475.86± 47.65 ○
optdigits	1070.51± 374.59	4527.79± 323.36 ○	1512.29± 342.05 ○	1812.11± 159.67 ○	1643.06± 159.60 ○	1049.54±178.75	890.14±175.62

○, ● statistically significant improvement or degradation

By contrast, Random Search generally consumes more time for larger datasets than GPO. It shows a distinctive trend: GPO reveals its advantage with the increasing size of datasets. This is an attractive point for GPO, which spends a comparatively short amount of time when applied to larger datasets.

4.2.2 Classification Prediction Based on C4.5

The decision tree learning algorithm C4.5 is implemented as J48 in WEKA, and several datasets will be experimented with using this software to gain insight into the effectiveness of the GPO algorithm when building decision trees. Here we adopt J48 as the base algorithm to optimize the parameters of C4.5 decision trees using the GPO, Grid Search and Random Search algorithms respectively. There exist two principal parameters of concern, which are MinNumObj denoted as M, and ConfidenceFactor denoted as C. M determines the minimum number of instances per leaf, for which the default value is 2. As discussed earlier, the C4.5 algorithm divides the dataset on attributes according to the greatest information gain. This process continues throughout the creation of the entire tree until the leaf nodes exhibit the minimum number of instances. If a leaf has less than the minimum number of examples from the dataset, the parent node and its children will be compressed into a single leaf node. If a dataset has a lot of noise, the setting value of M should be increased.

Table 4.19: Search Configuration for J48

	Configuration	Random Search	Grid Search	GPO
ConfidenceFactor	min	2^{-8}		
	max	2^{-1}		
	step	1 (exponent)		
MinNumObj	min	1		
	max	50		
	step	1		
Random Grid Locations		25,20,15,10,5		
GP γ				100
GP δ				0.01

C is the confidence threshold for pruning. The default confidence value is set at 25% and works reasonably well in most cases; however, if the actual error rate on a test set is much greater than the estimated error rate, it should be set to a lower value. The lower the confidence level, the more pruned the decision

tree.

Thus, the parameters to be optimized are M and C. The detailed configuration for the experiment can be seen in Table 4.19.

For all search algorithms, the search for C starts from 2^{-8} to 2^{-1}, increasing the exponent by 1 in each step, and the M increases to the maximum 50 from the minimum number 1. "Random grid locations" refers to the searching points for Random Search, in other words, that is how many points need to be randomly searched in total. This value is controllable. In this case, we set these values as 25, 20, 15, 10 and 5. There is a reason for these chosen numbers: 15 is the average number of grid locations evaluated using 2-fold cross-validation by the GPO algorithm in the specified datasets. The parameters γ as 100 and δ as 0.01 for the GP model applied in GPO proved to be the most appropriate ones in preceding experiments and are used again here.

We will again use the UCI datasets to analyse the performance of these three algorithms. Table 4.20 shows the percentage of correct classifications. There is no significant difference between Grid Search and GPO, which indicates that both are comparable in terms of accuracy. The results for Random Search are not as good as those for the other two algorithms in some cases. This is especially serious for some datasets with larger numbers of instances: for "letter", there is a nearly 2% difference. With decreasing random grid locations, fewer significant differences exist. The reason is that the implementation has to do more 10-fold cross-validation runs around the current "best" point to seek a relatively better one because fewer random grid locations are evaluated using 2-fold cross-validation. Take 5 random grid locations for example: although the experiment only randomly chooses 5 points to be 2-fold cross-validated at the initial stage, it will do multiple 10-fold cross-validation runs in sub-grids around the "best" point found in this initial stage. It will perform 10-fold cross-validation recursively until no better point is found. In short, GPO and Grid Search exhibit significant improvements over Random Search in terms of percent correct.

Table 4.21 clearly shows nearly all cases cost a large amount of time for searching, evaluating and cross-validation in Grid Search. GPO has a relatively large advantage in terms of time cost. The training time for Random Search

Table 4.20: Percent Correct of Three Methods for J48

Dataset	GPO	GridSearch	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	78.7333±16.29	78.3000±16.01	78.8333±16.77	77.9667±17.56	78.7667±17.76	78.7667±17.76	77.5000±16.35
zoo	93.8000± 7.08	93.2000± 7.23	91.6364± 7.01	91.6364± 7.01	91.7273± 6.94	91.7273± 6.94	89.9364± 7.81
lymphography	75.6476± 9.60	74.9524± 9.87	75.5000± 9.72	75.5857± 9.68	75.0381±10.34	74.8381±10.57	74.7048± 9.51
iris	93.8667± 5.50	93.7333± 5.75	93.8000± 5.63	94.0000± 5.72	94.0000± 5.32	94.0000± 5.32	93.6667± 5.79
hepatitis	78.9833± 8.34	79.2500± 7.97	79.7500± 7.30	79.9417± 7.33	80.9667± 7.26	81.1000± 7.12	80.6583± 7.10
autos	83.2190± 8.70	83.0238± 9.12	71.9619±11.16 •	72.4929±11.30 •	71.9357±12.88 •	70.8643±13.28 •	71.5024±11.78 •
sonar	73.2405± 8.89	71.9333± 9.07	72.9500± 9.24	72.6643± 9.19	72.4762± 8.98	72.5238± 8.99	74.0095± 9.92
Glass	67.1494± 9.72	67.0065±10.04	67.4004± 9.44	67.2100± 9.65	67.7056± 9.20	67.7078± 9.26	66.7987±10.26
audiology	83.3735± 8.08	82.7530± 8.16	71.9862± 7.01 •	72.1225± 7.07 •	71.2964± 5.91 •	71.2055± 5.96 •	74.4328± 9.17 •
heart-statlog	77.8889± 7.82	80.3704± 6.93	80.7407± 8.29	80.5556± 8.30	79.3333± 8.52	79.4815± 8.50	79.7407± 8.18
breast-cancer	71.9458± 6.20	73.0616± 6.28	72.5382± 6.12	72.0480± 6.00	72.3300± 6.00	72.3300± 6.00	71.7993± 5.89
hungarian-14-heart-disease	80.6667± 7.66	80.4954± 7.69	80.7678± 7.41	80.6678± 7.37	80.6724± 7.37	80.5690± 7.36	80.5690± 7.35
cleveland-14-heart-disease	75.3086± 6.94	76.0774± 6.79	75.5806± 7.08	75.6140± 7.16	75.0581± 7.01	75.0581± 7.01	74.3914± 6.85
ecoli	81.2870± 5.71	81.0187± 5.92	80.8503± 5.99	80.9394± 5.90	81.0526± 5.89	81.0517± 5.88	80.0027± 5.71
primary-tumor	41.1854± 6.77	40.0945± 6.84	40.7995± 6.55	40.8877± 6.46	40.8012± 6.47	40.6845± 6.35	40.1230± 6.71
ionosphere	89.2079± 4.31	89.2056± 4.84	89.0627± 4.95	89.2349± 4.81	88.6952± 4.79	88.7230± 4.80	88.8341± 4.50
horse-colic.ORIG	65.2830± 4.57	64.8401± 5.37	64.1914± 5.08	64.3859± 5.21	63.8664± 4.95	63.9474± 4.95	66.0068± 4.38
horse-colic	85.2327± 5.76	85.3431± 5.85	85.6952± 5.69	85.5863± 5.83	85.4520± 5.76	85.5060± 5.77	85.3716± 5.87
vote	95.6993± 2.78	95.8124± 2.68	95.3557± 2.88	95.5851± 2.74	95.5851± 2.74	95.6084± 2.73	95.6316± 2.76
arrhythmia	69.7947± 5.91	69.4483± 5.95	69.7541± 5.94	69.5560± 5.58	69.6691± 5.55	69.6464± 5.41	69.7116± 5.60
balance-scale	78.2527± 3.89	78.2355± 3.81	77.4531± 3.97	77.5486± 3.93	76.6695± 4.53	76.6372± 4.56	77.0335± 4.33
soybean	92.0205± 3.23	92.2549± 3.03	89.3416± 3.22 •	89.3416± 3.22 •	89.3122± 3.21 •	89.3122± 3.21 •	89.7513± 3.61 •
credit-rating	84.8841± 3.98	85.7826± 3.76	85.2174± 3.81	85.1884± 3.83	85.2899± 3.99	85.2754± 4.00	85.4203± 3.98
wisconsin-breast-cancer	94.5642± 2.72	94.6083± 2.66	94.4068± 2.77	94.3642± 2.81	94.2925± 2.83	94.2783± 2.84	94.3930± 2.78
pima-diabetes	74.9581± 4.64	74.6059± 4.69	74.5697± 4.91	74.8288± 4.93	74.8018± 4.83	74.8411± 4.83	74.5289± 4.81
vehicle	72.0709± 4.41	71.9374± 4.18	71.7497± 4.22	71.5497± 4.37	71.2424± 4.58	71.3010± 4.61	70.9220± 4.72
anneal.ORIG	93.5864± 2.48	93.6527± 2.48	92.0508± 3.17	92.0508± 3.17	90.0360± 3.71 •	90.0360± 3.71 •	90.3352± 3.55 •
anneal	99.2993± 0.90	99.2548± 0.92	98.3743± 1.40 •	98.4192± 1.36 •	98.0404± 1.53 •	98.0404± 1.53 •	96.7161± 1.73 •
vowel	82.6566± 4.17	82.7576± 4.15	78.0000± 5.34 •	78.0000± 5.34 •	78.0303± 5.43 •	78.0303± 5.43 •	81.1111± 4.57 •
german-credit	72.3000± 4.06	72.2000± 3.83	71.7100± 3.27	71.8000± 3.21	72.0600± 3.67	72.0600± 3.67	72.2100± 3.68
segment	96.8874± 1.27	96.8615± 1.24	96.1515± 1.42 •	96.1861± 1.40 •	96.3506± 1.37	96.3463± 1.39	96.2121± 1.40
splice	94.0000± 1.30	94.0157± 1.29	94.0721± 1.37	94.0721± 1.37	94.0940± 1.36	94.1034± 1.35	93.7712± 1.45
kr-vs-kp	99.3399± 0.36	99.4463± 0.37	99.3055± 0.40	99.3055± 0.40	99.3274± 0.46	99.3149± 0.46	99.2461± 0.39
hypothyroid	99.5229± 0.33	99.5388± 0.33	99.4513± 0.32	99.4539± 0.33	99.4619± 0.33	99.4645± 0.32	99.4301± 0.34
sick	98.6850± 0.58	98.7646± 0.58	98.6532± 0.68	98.6479± 0.68	98.5153± 0.76	98.5047± 0.76	98.5763± 0.72
waveform	76.3680± 1.92	76.4660± 1.80	76.5200± 1.87	76.4580± 1.86	76.4140± 1.80	76.3920± 1.79	76.5800± 1.78
optdigits	90.6228± 1.22	90.6548± 1.22	90.3559± 1.30	90.3559± 1.30	90.3772± 1.29	90.3772± 1.29	90.2936± 1.21
mushroom	100.0000± 0.00	100.0000± 0.00	100.0000± 0.00	100.0000± 0.00	100.0000± 0.00	100.0000± 0.00	100.0000± 0.00
pendigits	96.6448± 0.56	96.6484± 0.57	96.2018± 0.67 •	96.2018± 0.67 •	96.2118± 0.66 •	96.2118± 0.66 •	96.6002± 0.55
letter	88.5205± 0.83	88.5240± 0.83	86.8345± 0.86 •	86.8345± 0.86 •	86.8345± 0.86 •	86.8345± 0.86 •	88.4270± 0.83
Average	83.6674	83.6532	82.7393	82.7322	82.5947	82.5675	82.6737

◦, • statistically significant improvement or degradation

depends on the number of random grid locations as well as the size of the datasets. Random search for smaller datasets is usually distinctly better than GPO and Grid Search whereas for larger datasets it is evidently worse than GPO, which mainly depends on the 10-fold cross-validation times. Overall, these results indicate that GPO can work very well on larger datasets compared with Grid Search and Random Search.

Note that, at the beginning, the system will perform 2-fold cross-validation for every algorithm. The number of points to be searched can be figured out in Grid Search through simple calculation in advance; the number of search points for random search is fixed; GPO is the only one for which the number of search points is different for each dataset: it depends on the stopping criterion. For example, in this experiment, Grid Search will perform 400 2-fold cross-validations for all the UCI datasets, while this process does not stop in GPO until the stopping requirement is met, as shown in Table 4.22. Grid Search is much worse than the other two algorithms in this comparison.

As always in the implementations used in this thesis, once the "best" point is found in the preceding process and it is not in a border position, the software will perform multiple 10-fold cross-validation at locations surrounding this point and at the point itself. Table 4.23 compares the number of runs of 10-fold cross-validation among GPO, Grid Search and Random Search, which shows GPO improves significantly on Random Search, but not on Grid Search.

Overall, the GPO algorithm works well for C4.5 decision tree learning as we have seen through the analysis of the accuracy of the tree classification, experiment cross-validation runs and training time. This optimization method makes it possible to save a large amount of time when exploring the optimal parameters compared to trying every possible pair of values exhaustively. In addition, the accuracy obtained by its few searches is comparable with the one obtained through the whole grid search. The nature of the GPO algorithm is to find the points with useful information, and ignore those points which do not much help on the predictions. The core of the optimization algorithm is to apply the Gaussian process with the expected improvement in the search process, and this appears to identify high-quality parameter values for C4.5 decision tree induction.

Table 4.21: Training Time of Three Methods for J48

Dataset	GPO	GridSearch	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	0.1613± 0.05	0.1783± 0.02	0.0455± 0.02 •	0.0466± 0.02 •	0.0283± 0.02 •	0.0280± 0.02 •	0.0454± 0.02 •
zoo	0.1552± 0.03	0.2130± 0.01 ◦	0.0363± 0.03 •	0.0333± 0.03 •	0.0284± 0.03 •	0.0285± 0.03 •	0.0823± 0.01 •
lymphography	0.1862± 0.07	0.2621± 0.03 ◦	0.0810± 0.03 •	0.0815± 0.04 •	0.0669± 0.05 •	0.0655± 0.05 •	0.0988± 0.04 •
iris	0.1509± 0.03	0.2403± 0.01 ◦	0.0509± 0.01 •	0.0404± 0.02 •	0.0185± 0.01 •	0.0176± 0.01 •	0.0468± 0.02 •
hepatitis	0.3368± 0.20	0.3295± 0.09	0.1320± 0.07 •	0.1191± 0.07 •	0.0608± 0.07 •	0.0633± 0.08 •	0.1078± 0.08 •
autos	0.1940± 0.06	0.7363± 0.05 ◦	0.4623± 0.19 ◦	0.4649± 0.21 ◦	0.5349± 0.28 ◦	0.5192± 0.28 ◦	0.6927± 0.21 ◦
sonar	1.3940± 1.13	2.3790± 0.68 ◦	1.2654± 0.99	1.2828± 1.11	1.0679± 1.15	1.0353± 1.12	1.9945± 0.97
Glass	0.3918± 0.25	0.7881± 0.20 ◦	0.3377± 0.21	0.3298± 0.21	0.3342± 0.26	0.3220± 0.25	0.5612± 0.20
audiology	0.2204± 0.07	0.6489± 0.06 ◦	0.1490± 0.16	0.1402± 0.16	0.0773± 0.13 •	0.0744± 0.12 •	0.5510± 0.18 ◦
heart-statlog	0.3604± 0.22	0.6801± 0.16 ◦	0.3153± 0.12	0.3358± 0.15	0.3437± 0.22	0.3488± 0.22	0.3873± 0.19
breast-cancer	0.3381± 0.15	0.2844± 0.03	0.0680± 0.04 •	0.0599± 0.04 •	0.0369± 0.05 •	0.0386± 0.05 •	0.0363± 0.04 •
hungarian-14-heart-disease	0.3216± 0.17	0.5767± 0.16 ◦	0.2425± 0.17	0.1457± 0.13 •	0.1215± 0.14 •	0.1299± 0.16 •	0.1790± 0.13 •
cleveland-14-heart-disease	0.2957± 0.21	0.6186± 0.10 ◦	0.2200± 0.10	0.2190± 0.12	0.1810± 0.15	0.1811± 0.15	0.2287± 0.19
ecoli	0.3322± 0.22	0.8276± 0.17 ◦	0.2826± 0.18	0.2818± 0.19	0.2469± 0.20	0.2396± 0.19	0.4234± 0.14
primary-tumor	0.4209± 0.22	0.8162± 0.16 ◦	0.2747± 0.14	0.2752± 0.14	0.2631± 0.16	0.2581± 0.16	0.2847± 0.11
ionosphere	1.6810± 1.32	3.7182± 1.00 ◦	2.1316± 1.13	1.9524± 1.22	1.4035± 1.45	1.4085± 1.46	2.7133± 0.95 ◦
horse-colic.ORIG	0.4750± 0.18	0.7852± 0.10 ◦	0.1910± 0.14 •	0.1412± 0.13 •	0.0918± 0.12 •	0.1014± 0.14 •	0.1812± 0.11 •
horse-colic	0.4024± 0.23	0.7574± 0.18 ◦	0.2568± 0.16	0.3160± 0.19	0.1588± 0.19 •	0.1603± 0.19 •	0.2305± 0.09 •
vote	0.4524± 0.17	0.4076± 0.02	0.1536± 0.05 •	0.0787± 0.02 •	0.0324± 0.03 •	0.0358± 0.04 •	0.0100± 0.00 •
arrhythmia	25.8537±16.60	61.6450±20.60 ◦	30.9694±16.53	31.3048±17.14	29.5062±19.10	29.3250±19.25	35.0947±13.76
balance-scale	0.2714± 0.16	0.8702± 0.11 ◦	0.3324± 0.10	0.3160± 0.11	0.3774± 0.18	0.3702± 0.18	0.5046± 0.13 ◦
soybean	0.3072± 0.07	1.7910± 0.11 ◦	0.2473± 0.30	0.2236± 0.30	0.1786± 0.29	0.1787± 0.30	1.0743± 0.39 ◦
credit-rating	0.8525± 0.48	1.7140± 0.38 ◦	0.6728± 0.41	0.4940± 0.39	0.2176± 0.35 •	0.2195± 0.35 •	0.2596± 0.44 •
wisconsin-breast-cancer	0.3393± 0.23	1.1466± 0.18 ◦	0.3948± 0.21	0.3995± 0.24	0.3459± 0.28	0.3390± 0.28	0.6171± 0.18 ◦
pima-diabetes	1.0480± 0.75	2.2672± 0.44 ◦	0.8328± 0.70	0.7390± 0.73	0.5170± 0.78	0.5520± 0.82	0.8590± 0.67
vehicle	1.3692± 1.27	5.7860± 1.38 ◦	2.4606± 1.18	2.4518± 1.23	2.2201± 1.38	2.2086± 1.37	2.8175± 1.09 ◦
anneal.ORIG	0.3421± 0.15	2.8997± 0.32 ◦	1.4376± 0.31 ◦	1.3987± 0.31 ◦	1.2117± 1.15 ◦	1.2103± 1.14 ◦	2.9102± 0.83 ◦
anneal	0.4405± 0.30	2.7580± 0.09 ◦	1.4266± 0.63 ◦	1.3329± 0.65 ◦	1.3063± 0.83 ◦	1.3107± 0.84 ◦	1.7129± 0.43 ◦
vowel	0.6351± 0.02	9.8940± 0.22 ◦	2.7751± 3.04 ◦	2.6510± 3.04	2.3958± 3.13	2.3941± 3.13	8.8120± 1.04 ◦
german-credit	0.5275± 0.32	2.2072± 0.42 ◦	0.5467± 0.48	0.4476± 0.48	0.1852± 0.31 •	0.1850± 0.31 •	0.4495± 0.96
segment	1.6002± 1.78	21.7399± 2.17 ◦	6.4908± 4.61 ◦	6.2203± 4.63 ◦	6.3213± 5.25 ◦	6.2921± 5.21 ◦	11.5514± 3.56 ◦
splice	4.8469± 3.52	16.8986± 3.46 ◦	7.5644± 1.88 ◦	7.3998± 1.89	8.1373± 1.31 ◦	8.1332± 1.31 ◦	11.0231± 2.80 ◦
kr-vs-kp	0.4351± 0.23	5.4661± 0.27 ◦	2.2978± 0.81 ◦	2.2130± 0.80 ◦	2.2153± 0.90 ◦	2.1903± 0.89 ◦	3.2226± 0.38 ◦
hypothyroid	0.9780± 0.82	8.6403± 1.11 ◦	1.3142± 1.28	1.2306± 1.35	1.0145± 1.31	0.9951± 1.26	4.2934± 1.35 ◦
sick	1.6176± 1.88	9.9218± 1.65 ◦	4.4433± 2.32 ◦	4.2940± 2.29 ◦	4.1300± 3.30 ◦	3.9797± 3.14 ◦	7.6816± 2.32 ◦
waveform	51.0969±34.90	193.5207±27.65 ◦	58.4232±28.51	53.3193±27.62	49.0623±29.41	48.1476±28.73	59.7686±17.07
optdigits	14.2063±16.78	155.5311±22.00 ◦	38.7172±40.17	36.7620±40.03	34.3247±40.00	34.2831±39.94	92.4352±29.01 ◦
mushroom	0.8923± 0.31	5.4842± 0.18 ◦	1.0513± 0.32	0.9694± 0.34	0.2767± 0.01 •	0.2764± 0.01 •	0.9951± 0.34
pendigits	5.4501± 0.70	113.4177± 3.44 ◦	29.0186±28.10 ◦	27.5096±28.01 ◦	26.5354±29.33 ◦	26.7708±29.62 ◦	76.4136± 4.48 ◦
letter	17.7414± 1.39	301.9679± 9.57 ◦	32.9615±56.13	30.0495±60.03	25.4555±57.80	25.5519±58.16	358.9030±40.19 ◦
Average	3.4780	23.5204	5.7769	5.4518	5.0258	4.9992	17.2563

◦, • statistically significant improvement or degradation

Table 4.22: Number of 2-Fold Cross-validation Runs of Three Methods for J48

Dataset	GPO	GridSearch	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	11.4100± 3.45	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
zoo	10.0700± 0.70	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
lymphography	11.9300± 3.62	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
iris	10.3600± 1.31	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
hepatitis	18.9200±10.76	400.00±0.00 ○	25.00±0.00	20.00±0.00	15.00±0.00	15.00±0.00	5.00±0.00 ●
autos	10.2000± 0.67	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
sonar	14.3900± 4.78	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
Glass	13.4200± 4.33	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
audiology	10.3800± 1.20	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
heart-statlog	15.3800± 6.24	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
breast-cancer	20.3400± 8.48	400.00±0.00 ○	25.00±0.00	20.00±0.00	15.00±0.00	15.00±0.00	5.00±0.00 ●
hungarian-14-heart-disease	16.1700± 7.62	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00	15.00±0.00	15.00±0.00	5.00±0.00 ●
cleveland-14-heart-disease	15.1800± 8.64	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00	15.00±0.00	15.00±0.00	5.00±0.00 ●
ecoli	12.7900± 3.97	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
primary-tumor	15.3400± 5.75	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
ionosphere	14.9300± 5.43	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
horse-colic.ORIG	22.6200± 9.23	400.00±0.00 ○	25.00±0.00	20.00±0.00	15.00±0.00 ●	15.00±0.00 ●	5.00±0.00 ●
horse-colic	13.7300± 5.02	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
vote	28.4700± 8.81	400.00±0.00 ○	25.00±0.00	20.00±0.00 ●	15.00±0.00 ●	15.00±0.00 ●	5.00±0.00 ●
arrhythmia	15.7500± 4.75	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
balance-scale	12.1500± 3.69	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
soybean	12.2300± 2.34	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
credit-rating	28.6900±10.30	400.00±0.00 ○	25.00±0.00	20.00±0.00 ●	15.00±0.00 ●	15.00±0.00 ●	5.00±0.00 ●
wisconsin-breast-cancer	13.4700± 5.21	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
pima-diabetes	20.7600± 7.67	400.00±0.00 ○	25.00±0.00	20.00±0.00	15.00±0.00 ●	15.00±0.00 ●	5.00±0.00 ●
vehicle	14.4400± 5.05	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
anneal.ORIG	10.5100± 1.28	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
anneal	10.1300± 0.71	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
vowel	10.0400± 0.24	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
german-credit	17.2400± 6.72	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00	15.00±0.00	15.00±0.00	5.00±0.00 ●
segment	12.0800± 3.16	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
splice	13.2100± 3.85	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
kr-vs-kp	10.7700± 1.70	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
hypothyroid	18.4700± 5.06	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00	15.00±0.00	15.00±0.00	5.00±0.00 ●
sick	20.5600± 6.59	400.00±0.00 ○	25.00±0.00	20.00±0.00	15.00±0.00 ●	15.00±0.00 ●	5.00±0.00 ●
waveform	19.3600± 6.59	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00	15.00±0.00	15.00±0.00	5.00±0.00 ●
optdigits	13.3900± 3.73	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00	15.00±0.00	5.00±0.00 ●
mushroom	22.7700± 2.92	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ●	15.00±0.00 ●	15.00±0.00 ●	5.00±0.00 ●
pendigits	11.2200± 2.12	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
letter	10.8700± 1.18	400.00±0.00 ○	25.00±0.00 ○	20.00±0.00 ○	15.00±0.00 ○	15.00±0.00 ○	5.00±0.00 ●
Average	15.1035	400.00	25.00	20.00	15.00	15.00	5.00

○, ● statistically significant improvement or degradation

Table 4.23: Number of 10-Fold Cross-validation Runs of Three Methods for J48

Dataset	GPO	GridSearch	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	3.1500± 6.66	1.30± 4.53	11.78± 6.70 ○	12.95± 8.85 ○	6.95± 9.44	6.78± 9.25	16.11± 7.63 ○
zoo	4.29± 5.75	0.00± 0. ●	3.79± 6.92	3.79± 6.92	3.61± 6.89	3.61± 6.89	22.83± 2.63 ○
lymphography	3.89± 7.24	2.51± 5.75	11.50± 6.86 ○	12.34± 8.38 ○	10.08±10.11	9.90± 9.87	19.12± 7.32 ○
iris	2.96± 5.54	0.12± 1.20	8.29± 4.48 ○	7.68± 4.46	0.79± 3.55	0.79± 3.55	11.10± 4.60 ○
hepatitis	8.64±11.18	4.89±10.35	14.32± 8.19	12.56± 8.05	5.06± 9.03	5.54± 9.98	12.61±10.74
autos	0.27± 1.54	0.18± 1.27	14.01± 6.53 ○	14.37± 7.27 ○	17.11±10.01 ○	16.60± 9.95 ○	24.30± 6.67 ○
sonar	11.27±12.33	2.07± 5.57 ●	11.69± 9.82	11.68±10.60	9.21±10.09	8.96± 9.90	18.79± 6.78
Glass	5.48± 7.16	4.79± 6.40	9.91± 7.74	9.88± 7.73	10.36± 9.28	10.05± 9.16	19.95± 6.93 ○
audiology	0.36± 1.77	0.23± 1.66	3.90± 6.89	3.88± 6.86	1.67± 6.11	1.54± 5.68	24.54± 7.12 ○
heart-statlog	6.43± 9.11	4.02± 7.54	13.42± 6.27 ○	14.69± 7.98 ○	14.92±11.03	15.28±11.42	19.65±10.32 ○
breast-cancer	9.01±13.64	2.62± 5.75	11.88±12.75	11.14±13.57	4.28±11.32	5.±13.13	7.93±12.25
hungarian-14-heart-disease	6.58±11.69	6.65±14.34	18.23±16.79	10.79±11.20	7.49±11.56	8.53±14.15	14.21±12.65
cleveland-14-heart-disease	3.83± 6.58	2.83± 5.17	11.07± 6.76 ○	11.41± 8.36 ○	8.64± 9.15	8.64± 9.15	14.19±14.36
ecoli	5.40± 8.36	5.31± 7.10	10.47± 8.05	10.79± 8.71	9.56± 9.36	9.28± 9.10	19.15± 5.89 ○
primary-tumor	6.68± 6.55	8.09± 7.13	12.23± 7.36	12.30± 7.34	11.61± 8.06	11.64± 8.48	16.60± 6.29 ○
ionosphere	9.40± 9.08	3.24± 6.40	13.79± 7.31	13.12± 7.73	8.60±10.24	8.67±10.41	19.40± 6.80 ○
horse-colic.ORIG	8.41± 8.40	2.84± 5.61	9.05±11.13	5.57± 8.81	3.62± 8.86	4.42±10.89	12.76± 8.97
horse-colic	8.20±10.70	4.± 8.81	10.67± 9.06	13.78±10.99	5.48± 9.15	5.62± 9.50	10.94± 3.89
vote	4.50± 5.11	0.36± 1.77 ●	15.84± 6.61 ○	9.10± 1. ○	0.81± 3.16	1.07± 5.03	0.± 0. ●
arrhythmia	11.34± 8.49	9.66± 9.57	13.15± 8.02	13.40± 8.40	12.75± 9.19	12.69± 9.21	16.16± 7.45
balance-scale	2.40± 4.68	2.05± 4.23	11.79± 4.56 ○	11.51± 4.73 ○	14.20± 8.06 ○	13.99± 8.05 ○	23.08± 5.41 ○
soybean	0.09± 0.90	0.23± 1.66	1.75± 5.23	1.75± 5.23	1.48± 5.08	1.48± 5.08	19.50± 7.02 ○
credit-rating	9.25±12.24	4.91± 8.70	15.88±13.44	10.78±10.66	3.41± 8.83	3.43± 8.76	5.67±11.23
wisconsin-breast-cancer	3.64± 6.80	3.75± 6.06	11.12± 7.79 ○	11.71± 8.44 ○	10.37±10.21	10.06±10.01	21.52± 6.23 ○
pima-diabetes	14.51±17.45	5.93± 9.96	15.01±16.94	12.99±17.09	9.08±17.84	9.86±18.86	16.65±15.24
vehicle	5.09± 7.10	5.72± 7.61	12.17± 6.87 ○	12.40± 7.13 ○	11.44± 7.87	11.36± 7.77	16.46± 6.07 ○
anneal.ORIG	0.09± 0.90	0.41± 2.07	9.23± 2.14 ○	9.23± 2.14 ○	7.64± 8.07 ○	7.64± 8.07 ○	23.39± 5.67 ○
anneal	0.63± 2.31	0.± 0.	11.42± 5.87 ○	10.87± 6.16 ○	11.11± 7.91 ○	11.11± 7.91 ○	18.77± 4.32 ○
vowel	0.00± 0.00	0.00± 0.00	5.55± 8.23	5.55± 8.23	5.24± 8.45	5.24± 8.45	23.67± 2.86 ○
german-credit	3.88± 6.03	3.57± 5.84	5.61± 6.76	4.17± 6.47	1.12± 4.15	1.12± 4.15	8.3900±20.87
segment	0.78± 2.93	1.53± 4.00	9.31± 8.53 ○	9.27± 8.53 ○	9.92± 9.61 ○	9.87± 9.53 ○	20.75± 6.53 ○
splice	6.57± 6.36	7.10± 6.40	15.01± 3.25 ○	15.01± 3.25 ○	16.62± 1.80 ○	16.62± 1.80 ○	25.33± 5.57 ○
kr-vs-kp	0.27± 1.54	0.35± 2.03	14.68± 6.15 ○	14.68± 6.15 ○	15.18± 6.91 ○	15.04± 6.90 ○	24.55± 3.14 ○
hypothyroid	0.82± 4.26	0.97± 3.57	3.89± 7.06	4.06± 7.45	3.49± 7.18	3.39± 6.93	23.04± 5.73 ○
sick	2.23± 7.03	1.81± 4.02	11.39± 6.51 ○	11.35± 6.52 ○	11.54±10.03 ○	11.09± 9.67 ○	23.50± 5.48 ○
waveform	10.18± 8.58	12.07± 6.59	11.52± 6.79	10.83± 6.64	10.26± 7.07	10.04± 6.88	13.67± 4.17
optdigits	1.19± 3.34	1.90± 4.43	5.95± 8.37	5.95± 8.37	5.84± 8.34	5.84± 8.34	19.07± 5.96 ○
mushroom	1.53± 3.40	0.00± 0.00	7.31± 3.87 ○	7.34± 4.12 ○	0.00± 0.00	0.00± 0.00	10.47± 4.03 ○
pendigits	0.00± 0.00	0.09± 0.90	6.63± 8.73 ○	6.63± 8.73 ○	6.83± 9.19 ○	6.83± 9.19 ○	23.62± 1.25 ○
letter	0.00± 0.00	0.00± 0.00	0.74± 3.65	0.74± 3.65	0.74± 3.65	0.74± 3.65	23.90± 0.44 ○
Average	4.58	2.95	10.37	9.80	7.70	7.73	17.63

○, ● statistically significant improvement or degradation

4.2.3 Classification Prediction Based on LogitBoost

In this section, we will optimize a logistic regression model. The LogitBoost algorithm derives from the earlier AdaBoost algorithm and converts the exponential loss function of AdaBoost into minimizing the negative binomial log-likelihood function. On the one hand, this modification makes the Boosting algorithm less susceptible to noise. On the other hand, the AdaBoost algorithm has no tuning parameters with the exception of the number of boosting rounds while LogitBoost provides the important parameter "shrinkage". The default value of "shrinkage" is set to be 1.0 in WEKA, which means no shrinkage at all. However, smaller "shrinkage" is often regarded as a good choice for classification performance to prevent overfitting.

We will adopt trees as the base learner for LogitBoost. Lutz [31] states that trees used as the base classifier in the LogitBoost algorithm can easily "model different degrees of interaction" as well as "no variable transformations are needed" [31]. The depth of the tree learner is another parameter to be considered. Different depths will produce different predictive performance. As the depth of the tree should be pre-fixed before the experiment, we will provide the bounds on the tree depth in advance. The tree depth is initially set to be 1, and increases with step size 1 to the max depth specified by the user. In some cases, there exist different depths that have approximately the same performance, such as accuracy or error rate. Under this kind of situation, we will take the smaller tree to avoid a complex and big model.

We have talked about the two parameters for the boosted trees so far: the shrinkage ν of the LogitBoost algorithm itself and the maximum depth d_{max} of the tree in the base classifier. Given the range of each parameter respectively, we thus form a grid with a number of points. Every point in the gird is a pair of parameter settings, which we can apply in the experiment to build the predictive model. The outcome of the model varies with the different pairs of points. We should find the point with the "best" performance amongst these points. This point corresponds to the optimal parameters, which is used for building the model to classify the test data points.

Grid Search is a possible search method to find the optimum point in this problem. It does a full search of the search area and evaluates all possible

points to decide the value of the optimum parameters. The main drawback of this approach is that it may consume a large amount of time on useless points. That would not have any effect on the final result even if we deleted them. Consequently, the GPO algorithm becomes an alternative algorithm to make up for the short comings in Grid Search, which not only provides comparable performance, but also quickly jumps into promising areas if the search space. This optimization approach helps to reduce the pressure when running a large dataset from the viewpoint of computational cost. Random Search, which randomly chooses points in the parameter space is another low-cost alternative. The quality of this stochastic search method depends on the number of points to be chosen. Hence, it is very important to specify the number of the random grid locations appropriately in advance.

Before running the experiment to optimize the parameters of the Logit-Boost algorithm, we should give the experimental set up. As stated, we will optimize the pair of parameters (ν, d_{max}) that have a decisive role in determining the quality of the model. In terms of the shrinkage parameter ν , we specify a wide range, from the minimum value 0.1 to the maximum value 1 with increment 0.1. It makes no sense to use values higher than 1 for this parameter. The other parameter d_{max} refers to the maximum depth of the tree learner. According to practical experience, we set its value between 1 and 5. Hence, there are 10 $\times$ 5 grid locations in the parameter space. This is a relatively smaller grid than the grids of the preceding algorithms. As a result, it is not a huge task to search through all the locations of the grid using Grid Search. Hence, it appears that the GPO algorithm does not have a clear advantage to outperform Grid Search approach in this application. With regards to Random Search, the performance and the training time varies with the number of grid locations specified by the user. Generally, more points mean a higher probability of choosing a good point as every point in the grid has the same opportunity to be chosen.

We will give comparative data on accuracy of the model, training time and number of cross-validation runs amongst Grid Search, GPO and Random Search, see Tables 4.24 to 4.27. In these tables, the GPO algorithm is always regarded as the baseline algorithm. We will present the merits and drawbacks of this optimization method from various points of view. Table 4.24 provides

Table 4.24: Percent Correct of Three Methods for LogitBoost

Dataset	GPO	Grid Search	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	86.87±12.61	87.13±15.36	85.87±14.31	86.13±14.24	86.80±12.63	86.47±14.59	86.47±14.20
zoo	74.11± 8.91	74.11± 8.91	74.11± 8.91	74.11± 8.91	74.11± 8.91	74.91± 6.27	72.51±12.47
lymphography	82.99± 9.10	82.70± 8.72	82.71± 8.86	82.02± 8.87	81.76± 9.69	81.87±10.23	82.14± 9.42
iris	94.13± 5.81	93.87± 6.00	94.53± 5.82	94.00± 5.59	94.80± 5.44	94.40± 5.61	94.13± 5.33
hepatitis	84.39± 9.10	83.50± 8.42	84.82± 8.87	83.89± 8.52	84.54± 8.49	84.29± 8.30	85.07± 8.01
Glass	79.47± 6.86	79.47± 6.55	79.26± 6.41	79.45± 6.35	79.73± 6.47	79.61± 6.25	79.45± 6.51
audiology	84.56± 7.74	83.47± 7.73	83.75± 7.92	83.66± 7.15	82.44± 7.64	82.34± 7.62	83.40± 8.06
heart-statlog	82.81± 6.11	82.74± 6.11	80.22± 6.72	80.00± 6.65	79.56± 6.95	79.26± 7.22	79.56± 5.90
breast-cancer	69.38± 7.22	68.76± 7.98	68.62± 7.72	68.68± 7.72	67.22± 7.11	68.07± 7.54	68.27± 7.41
hungarian	82.05± 6.96	82.04± 6.98	81.29± 7.18	81.01± 6.86	79.37± 7.04	78.83± 6.88	80.20± 6.80
cleveland	82.38± 7.05	81.72± 7.24	81.12± 7.22	80.60± 7.26	78.87± 7.82	79.33± 7.37	79.98± 6.98
ecoli	83.34± 5.49	82.98± 5.26	83.29± 5.69	83.41± 5.91	83.71± 5.82	83.64± 5.51	83.94± 5.76
primary-tumor	48.33± 5.57	48.33± 5.57	46.72± 6.06	46.84± 6.07	43.60± 6.47 •	43.95± 7.02	41.95± 6.94 •
ionosphere	93.28± 3.97	93.05± 4.31	93.51± 3.85	93.40± 4.03	93.17± 4.11	93.17± 4.11	93.23± 4.26
horse-colic.ORIG	67.33± 4.17	67.01± 4.55	67.98± 3.47	67.98± 3.47	68.15± 3.30	68.15± 3.30	68.15± 3.30
horse-colic	82.82± 6.22	82.77± 6.03	82.83± 5.96	82.94± 5.92	84.02± 5.79	82.98± 6.09	82.72± 5.74
vote	96.04± 3.21	95.82± 3.22	95.86± 3.10	96.00± 3.09	95.08± 3.68	95.13± 3.64	94.99± 3.40
arrhythmia	73.55± 6.17	73.24± 5.40	72.84± 5.40	72.97± 5.07	73.02± 5.44	73.24± 5.18	72.71± 5.27
balance-scale	94.34± 1.62	94.53± 1.59	94.59± 1.74	94.72± 1.69	92.77± 0.93 •	92.67± 0.86 •	92.67± 0.86 •
soybean	93.70± 2.62	93.70± 2.62	93.61± 2.87	93.64± 2.90	92.27± 2.97	92.38± 2.80	92.38± 2.94
credit-rating	86.46± 3.82	86.46± 3.76	84.99± 3.89	85.39± 3.93	84.64± 3.99	84.72± 3.96	84.84± 4.14
wisconsin	95.99± 2.04	95.91± 2.20	96.20± 1.88	96.17± 1.84	96.34± 1.92	96.31± 2.08	96.28± 2.06
pima-diabetes	75.80± 4.62	75.83± 4.45	75.68± 4.61	75.68± 4.58	74.95± 4.76	74.92± 4.78	74.66± 4.68
vehicle	77.75± 4.48	77.90± 3.96	77.83± 4.27	78.19± 4.13	78.09± 4.25	78.21± 4.48	78.13± 4.64
anneal	99.51± 1.01	99.49± 1.01	99.60± 0.83	99.67± 0.68	99.64± 0.79	99.56± 0.81	99.60± 0.80
anneal.ORIG	96.46± 2.13	96.59± 1.90	96.50± 2.16	96.66± 2.09	96.33± 2.06	96.10± 2.28	96.21± 2.17
german-credit	75.80± 4.19	75.74± 3.82	75.20± 3.81	75.30± 4.04	74.38± 4.16	74.34± 3.92	74.30± 3.79
segment	98.76± 0.59	98.79± 0.55	98.66± 0.64	98.69± 0.60	98.67± 0.65	98.68± 0.64	98.68± 0.66
splice	52.58± 0.46	52.58± 0.46	52.58± 0.46	52.58± 0.46	52.58± 0.46	52.58± 0.46	52.58± 0.46
kr-vs-kp	99.70± 0.34	99.71± 0.32	99.71± 0.32	99.72± 0.32	99.72± 0.32	99.72± 0.30	99.75± 0.28
sick	99.05± 0.49	99.06± 0.51	99.08± 0.50	99.09± 0.51	99.09± 0.51	99.10± 0.50	99.05± 0.50
hypothyroid	99.61± 0.35	99.60± 0.36	99.63± 0.34	99.63± 0.33	99.60± 0.34	99.60± 0.33	99.59± 0.33
waveform	85.37± 1.47	85.34± 1.46	85.26± 1.37	85.28± 1.30	84.92± 1.84	84.93± 1.79	85.17± 1.76
mushroom	100.00± 0.00	100.00± 0.00	100.00± 0.00	100.00± 0.00	100.00± 0.00	100.00± 0.00	100.00± 0.00

○, • statistically significant improvement or degradation

the accuracy using these three methods for a number of datasets from the UCI repository. The accuracy of the GPO algorithm and the Grid Search algorithm are shown in the first and the second column of this table respectively. We find that the GPO algorithm is not inferior to Grid Search in terms of accuracy. In many cases, the GPO algorithm presents slightly better performance. The rest of the table records the accuracy when employing the Random Search algorithm with non-identical random grid locations. Despite Random Search being a very simple search method without a regular pattern, it again shows surprisingly good performance. A possible reason is that Random Search just pays attention to the "best" point amongst the chosen points and never consider the others. However, it is possible that it acquires inappropriate parameter values because the system misses a good point, such as on the "primary-tumor" dataset. The accuracy on this dataset is significantly lower than for other approaches when we specify the number of random grid locations as 10. This phenomenon will occur when the process falls into a poor area. Although Random Search performs very well in some cases, it does not guarantee that it succeeds in finding the "best" parameter combinations.

The training time is not only an important standard to measure the quality of an algorithm, but also plays a crucial role in the cost of computation. Researchers always try to reduce the time cost without affecting performance. From this standard point of view, GPO is a good method in comparison with Grid Search. Grid Search is a time-consuming process which cross-validates and evaluates every point along its search path. In contrast, GPO only picks the valuable points and jumps over useless points, which achieves the goal of saving time. Table 4.25 gives good evidence to show that GPO performs better than Grid Search in terms of training time for most of the datasets.

The table also provides the training time with respect to Random Search for different numbers of grid locations. The training time always shows obvious differences for different numbers of grid locations. Fewer grid locations require a larger amount of time in Random Search, which implies that Random Search with fewer grid locations puts more effort on advanced search based on 10-fold cross-validation. It also implies that the initial search does not take up a great deal of time. Note that, in order to reduce time spent on the advanced search,

Table 4.25: Training Time of Three Methods for LogitBoost

Dataset	GPO	Grid Search	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	2.85± 3.48	3.64± 2.43	3.18± 3.14	3.02± 3.09	4.49± 2.64	4.15± 3.44	4.63± 2.75
zoo	16.69± 10.97	14.96± 0.54	17.73± 10.13	8.41± 6.16	19.57± 8.08	8.86± 11.75	19.75± 9.80
lymphography	4.20± 4.67	8.23± 4.51 ○	5.99± 4.96	5.57± 5.47	11.07± 6.95 ○	9.19± 7.29	10.10± 6.23 ○
iris	2.84± 2.68	2.27± 1.72	1.89± 2.06	1.52± 1.86	2.90± 1.75	2.53± 2.80	2.88± 1.91
hepatitis	4.08± 4.81	8.93± 6.01	5.77± 5.99	6.23± 6.30	11.04± 5.33 ○	8.59± 5.81	9.82± 4.68 ○
Glass	12.91± 13.57	21.61± 13.12	15.85± 12.92	15.77± 12.68	14.51± 11.72	13.88± 13.22	20.93± 10.85
audiology	53.30± 60.11	155.56± 95.27 ○	82.16± 75.69	133.97± 138.05	158.12± 125.93 ○	123.26± 126.43	126.07±108.26
heart-statlog	3.21± 4.57	8.39± 4.39 ○	7.05± 6.74	6.64± 7.38	9.48± 7.74 ○	7.81± 7.21	12.12± 8.17 ○
breast-cancer	1.86± 2.18	5.82± 2.95 ○	3.32± 3.00	3.18± 3.06	5.83± 4.72 ○	5.89± 4.18 ○	6.48± 3.65 ○
hungarian	5.64± 6.84	16.99± 10.06 ○	10.26± 10.42	9.84± 11.11	20.18± 13.39 ○	17.28± 13.62 ○	20.02± 10.66 ○
cleveland	5.69± 9.56	14.96± 11.53 ○	10.28± 9.70	9.85± 10.79	14.24± 12.53	12.15± 11.36	17.73± 10.18 ○
ecoli	8.58± 13.91	25.09± 20.69 ○	24.78± 21.69	21.84± 19.38	24.71± 16.66 ○	19.23± 18.61	29.90± 15.30 ○
primary-tumor	13.53± 3.65	40.62± 9.72 ○	34.88± 31.47	23.03± 20.88	72.20± 60.44 ○	76.42± 57.69 ○	76.25± 49.06 ○
ionosphere	31.15± 24.52	47.02± 32.46	43.91± 32.10	41.89± 30.64	45.39± 22.11	39.88± 26.30	50.03± 23.23
horse-colic.ORIG	69.31± 48.30	46.04± 10.15	28.30± 13.13 ●	18.52± 3.87 ●	60.44± 13.54	9.33± 1.92 ●	49.36± 1.89
horse-colic	6.65± 7.51	24.41± 15.70 ○	14.10± 11.85	12.56± 12.81	17.32± 18.00	23.94± 20.59 ○	27.48± 18.51 ○
vote	3.93± 5.15	9.05± 5.91 ○	6.12± 5.08	4.41± 3.80	16.77± 13.11 ○	13.17± 12.39	18.07± 13.08 ○
arrhythmia	761.83± 860.82	1490.72±905.54	1239.57±1074.33	1123.70± 968.34	1509.26± 928.30	1464.16±1054.09	1776.03±736.07 ○
balance-scale	2.09± 0.20	8.85± 1.41 ○	5.35± 0.90 ○	4.16± 0.61 ○	12.88± 2.08 ○	12.21± 1.84 ○	11.14± 1.80 ○
soybean	36.42± 1.70	146.51± 36.41 ○	120.14± 88.49 ○	84.82± 91.00	344.78± 230.93 ○	285.04± 202.70 ○	404.84±184.29 ○
credit-rating	5.52± 4.81	21.50± 10.62 ○	16.03± 13.99 ○	15.95± 15.44	20.64± 20.00 ○	19.23± 20.28	26.54± 21.76 ○
wisconsin	13.13± 11.76	18.91± 12.13	19.02± 15.50	16.98± 14.34	19.09± 12.82	18.32± 16.55	24.81± 10.49 ○
pima-diabetes	6.59± 6.26	15.97± 9.66 ○	9.76± 7.16	9.24± 10.07	17.66± 11.99 ○	22.54± 14.30 ○	23.09± 13.45 ○
vehicle	86.29± 74.07	155.93± 87.69 ○	125.43± 80.46	112.20± 64.88	133.15± 74.90	102.20± 87.53	116.03± 50.61
anneal	92.32± 81.24	103.18± 78.47	78.36± 78.92	92.25± 116.90	141.54± 94.02	121.95± 126.86	132.54± 94.04
anneal.ORIG	60.09± 64.47	131.18± 74.98 ○	72.59± 55.05	82.65± 87.88	147.75± 129.74	225.80± 144.68 ○	182.07±105.48 ○
german-credit	18.27± 17.88	48.29± 23.29 ○	26.23± 20.29	24.98± 24.02	62.58± 38.82 ○	57.11± 40.15 ○	62.34± 39.31 ○
segment	258.96± 255.35	426.10±303.66 ○	441.47± 328.56	384.19± 278.90	398.71± 277.43	342.74± 311.38	413.70±257.68
splice	2894.86±2287.83	2020.79±918.57	4455.11±2693.34	2574.53±2550.95	5197.86±1761.90 ○	449.96± 177.43 ●	4042.40±456.47
kr-vs-kp	182.52± 160.87	357.80±194.37 ○	251.68± 197.92	308.88± 257.35	246.51± 194.36	209.93± 230.52	314.59±234.23
sick	232.24± 214.98	327.02±231.26	243.15± 245.88	200.77± 211.99	201.89± 220.95	180.74± 222.64	378.36±282.92
hypothyroid	187.25± 163.27	331.15±199.31 ○	215.28± 221.36	232.36± 265.75	457.43± 334.41 ○	503.16± 334.28 ○	530.55±272.52 ○
waveform	520.76± 450.63	909.35±482.80 ○	518.49± 437.09	418.00± 426.59	875.89± 984.94	722.39± 773.98	1263.74±748.51 ○
mushroom	264.92± 136.20	147.83± 37.11 ●	72.51± 17.81 ●	101.50± 100.67 ●	279.84± 67.79	143.72± 163.90 ●	267.57± 87.95

○, ● statistically significant improvement or degradation

Table 4.26: Number of 2-Fold Cross-validation Runs for LogitBoost

Dataset	GPO	GS	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	11.86±2.94	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
zoo	11.46±3.25	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
lymphography	12.78±3.54	50±0 ○	25±0 ○	20±0 ○	15±0	10±0 ●	5±0 ●
iris	198±6.81	50±0 ○	25±0 ○	20±0	15±0	10±0 ●	5±0 ●
hepatitis	12.70±4.23	50±0 ○	25±0 ○	20±0 ○	15±0	10±0	5±0 ●
Glass	12.82±4.23	50±0 ○	25±0 ○	20±0 ○	15±0	10±0	5±0 ●
audiology	12.70±3.73	50±0 ○	25±0 ○	20±0 ○	15±0	10±0	5±0 ●
heart-statlog	10.96±2.53	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
breast-cancer	11.50± 37	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
hungarian	11.22±2.55	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
cleveland	10.84±2.38	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
ecoli	12.50±4.46	50±0 ○	25±0 ○	20±0 ○	15±0	10±0	5±0 ●
primary-tumor	102±0.14	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
ionosphere	14.90±4.60	50±0 ○	25±0 ○	20±0 ○	15±0	10±0 ●	5±0 ●
horse-colic.ORIG	21.58±5.27	50±0 ○	25±0	20±0	15±0 ●	10±0 ●	5±0 ●
horse-colic	11.78±3.38	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
vote	13.10±4.80	50±0 ○	25±0 ○	20±0 ○	15±0	10±0	5±0 ●
arrhythmia	12.72±3.67	50±0 ○	25±0 ○	20±0 ○	15±0	10±0 ●	5±0 ●
balance-scale	10.46±1.22	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
soybean	102±0.14	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
credit-rating	10.70±1.88	50±0 ○	25±0 ○	20±0 ○	15±0 ○	10±0	5±0 ●
wisconsin	18.60±6.58	50±0 ○	25±0 ○	20±0	15±0	10±0 ●	5±0 ●
pima-diabetes	13.58±4.40	50±0 ○	25±0 ○	20±0 ○	15±0	10±0 ●	5±0 ●
vehicle	14.40±3.92	50±0 ○	25±0 ○	20±0 ○	15±0	10±0 ●	5±0 ●
anneal	23.16±3.97	50±0 ○	25±0	20±0 ●	15±0 ●	10±0 ●	5±0 ●
anneal.ORIG	156±5.54	50±0 ○	25±0 ○	20±0 ○	15±0	10±0 ●	5±0 ●
german-credit	13.84±3.67	50±0 ○	25±0 ○	20±0 ○	15±0	10±0 ●	5±0 ●
segment	21.18±3.40	50±0 ○	25±0 ○	20±0	15±0 ●	10±0 ●	5±0 ●
splice	24.28±1.92	50±0 ○	25±0	20±0 ●	15±0 ●	10±0 ●	5±0 ●
kr-vs-kp	19.18±4.46	50±0 ○	25±0 ○	20±0	15±0 ●	10±0 ●	5±0 ●
sick	23.26±4.46	50±0 ○	25±0	20±0 ●	15±0 ●	10±0 ●	5±0 ●
hypothyroid	24.94±2.91	50±0 ○	25±0	20±0 ●	15±0 ●	10±0 ●	5±0 ●
waveform	19.90±6.98	50±0 ○	25±0 ○	20±0	15±0	10±0 ●	5±0 ●
mushroom	24.50±1.42	50±0 ○	25±0	20±0 ●	15±0 ●	10±0 ●	5±0 ●

○, ● statistically significant improvement or degradation

the GPO algorithm tries to acquire the near optimal point in the initial search rather than finding it in the advanced search, and it generally appears to do so successfully.

Tables 4.26 and 4.27 show the 2-fold cross-validation runs and 10-fold cross-validation runs respectively. In Table 4.26, the amount of 2-fold cross-validation runs in Grid Search and Random Search is specified in the experimental set ups. By contrast, we have no idea how many 2-fold cross-validation runs will be performed by GPO before obtaining the results. We find the GPO algorithm only explores a small part of the parameter space. The results shown in Table

Table 4.27: Number of 10-Fold Cross-validation Runs for LogitBoost

Dataset	GPO	Grid Search	RS(25)	RS(20)	RS(15)	RS(10)	RS(5)
labor	6.50±10.37	4.68±7.08	5.38±8.31	5.68±8.38	11.54±7.67	10.04± 9.25	11.80±7.09
zoo	7.16± 5.73	0.00±0.00 •	5.90±6.13	1.54±4.04 •	8.98±4.71	3.12± 5.91	10.28±3.56
lymphography	3.10± 5.34	3.00±4.96	3.72±6.01	3.26±5.79	10.62±6.83 ○	8.02± 7.01	10.10±5.95 ○
iris	8.14±10.59	2.64±7.34	3.90±7.98	3.50±7.52	10.66±7.57	8.60±11.27	11.56±8.20
hepatitis	3.72± 6.44	4.44±6.47	3.04±6.05	3.78±6.14	10.92±5.67 ○	8.24± 6.33	10.10±4.77 ○
Glass	4.68± 6.62	5.60±6.56	5.08±6.51	5.98±6.56	6.48±6.60	5.80± 6.64	9.74±4.84
audiology	1.82± 4.04	3.68±5.67	3.04±5.37	3.60±5.76	6.80±6.22 ○	5.56± 6.39	6.94±5.61 ○
heart-statlog	1.18± 3.76	1.36±3.92	2.04±4.94	2.46±5.20	6.28±6.65	5.16± 5.74	8.16±4.38 ○
breast-cancer	0.90± 2.73	1.26±3.15	1.16±4.21	1.34±4.35	6.20±6.84	7.28± 6.22 ○	9.16±5.65 ○
hungarian	1.20± 3.37	2.46±4.93	1.48±4.66	1.94±5.07	8.70±6.98 ○	7.46± 6.90 ○	9.66±5.25 ○
cleveland	1.46± 5.19	1.64±5.29	1.94±4.70	2.26±5.09	5.82±6.79	4.98± 5.77	8.24±4.68 ○
ecoli	1.58± 5.12	4.12±7.26	6.12±7.14	6.30±7.10	8.72±6.96 ○	5.88± 6.31	10.60±5.47 ○
primary-tumor	0.00± 0.00	0.00±0.00	0.56±2.95	0.56±2.95	6.96±6.76 ○	7.48± 5.60 ○	7.94±4.83 ○
ionosphere	6.08± 6.24	5.88±6.37	7.28±7.41	8.08±7.34	10.84±6.17	8.86± 6.65	11.58±5.83
horse-colic.ORIG	5.04± 5.33	0.00±0.00 •	0.90±2.73 •	0.00±0.00 •	8.46±2.16	0.00± 0.00 •	9.00±0.00 ○
horse-colic	0.48± 2.45	2.04±5.09	1.06±3.28	1.06±3.28	4.12±6.60	6.88± 7.07 ○	9.30±6.75 ○
vote	1.58± 4.06	2.38±4.79	0.56±2.80	0.74±3.04	8.22±7.25 ○	7.24± 7.13	9.22±6.24 ○
arrhythmia	3.58± 5.85	3.84±5.79	4.94±6.56	5.00±6.36	8.90±6.41	8.18± 6.39	10.98±4.61 ○
balance-scale	0.00± 0.00	0.00±0.00	0.00±0.00	0.00±0.00	9.34±1.44 ○	9.00± 0.00 ○	9.00±0.00 ○
soybean	0.00± 0.00	0.00±0.00	0.52±2.58	0.52±2.58	9.06±6.50 ○	8.50± 6.38 ○	11.28±5.63 ○
credit-rating	0.18± 1.27	0.50±2.57	1.22±3.81	1.52±3.92	3.74±5.45	3.66± 5.33	6.20±5.47 ○
wisconsin	4.22± 5.69	4.10±6.13	5.64±7.00	5.88±7.01	8.30±6.97	6.42± 6.82	9.88±3.82 ○
pima-diabetes	1.50± 3.49	1.44±3.80	0.78±2.70	1.24±3.46	7.02±5.91 ○	8.60± 5.42 ○	9.68±4.81 ○
vehicle	5.62± 6.61	8.16±7.10	8.82±7.37	9.32±7.20	10.72±6.71	7.76± 6.92	10.88±5.21
anneal	7.26± 8.96	4.10±7.48	5.64±8.72	4.74±8.61	11.98±9.03	8.78±10.45	11.78±7.86
anneal.ORIG	0.96± 3.40	1.32±3.74	0.82±3.29	1.28±3.94	5.92±6.39 ○	8.54± 5.96 ○	8.86±5.44 ○
german-credit	2.68± 4.82	3.36±4.84	1.12±3.50	1.12±3.50	9.20±6.52 ○	8.22± 6.31	9.54±5.88 ○
segment	3.22± 5.80	3.84±5.77	6.92±6.76	7.10±6.69	7.88±6.61	6.34± 6.55	9.22±6.01 ○
splice	3.90± 4.70	0.00±0.00 •	7.56±5.63	3.98±5.43	10.18±3.59 ○	0.00± 0.00 •	9.00±0.00 ○
kr-vs-kp	4.44± 5.70	8.02±6.74	5.74±6.34	5.74±6.34	6.28±6.21	4.16± 5.63	7.32±5.52
sick	3.78± 6.03	4.62±6.47	3.28±6.16	3.60±6.40	3.84±6.49	3.54± 6.05	8.72±7.45
hypothyroid	2.00± 4.46	1.94±4.63	1.84±4.74	2.40±5.31	8.88±7.25 ○	9.78± 7.06 ○	11.34±5.83 ○
waveform	1.02± 3.13	1.08±3.31	1.00±3.49	1.00±3.49	4.38±6.55	4.52± 6.49	8.74±5.63 ○
mushroom	7.64± 4.43	0.00±0.00 •	0.00±0.00 •	1.26±3.15 •	9.00±0.00	3.06± 4.31 •	9.28±1.39

○, • statistically significant improvement or degradation

4.24 tell us its accuracy performance is comparable to Grid Search. Hence, the GPO algorithm is a promising optimization method, which applies a limited amount of time to obtain good performance. From Table 4.27, it is clear to see that GPO and Grid Search do not take many 10-fold cross-validation runs for most of datasets (except for a few special cases). This means the "centred" point found by the initial search is close to the optimal point. However, Random Search always runs many 10-fold cross-validations around the so-called "best" point collected so far. The reason is that the "best" point found in the initial search is not good enough so that the search process has to seek a better one in the parameter space. This situation becomes more serious when only a few points are chosen randomly.

4.3 Summary

Through the experiments above, we can see that the GPO algorithm achieves a comparably high performance, in conjunction with a significantly smaller number of cross-validation runs and much reduced training time compared to the Grid Search algorithm, both in regression problems and in classification problems. In some cases, Random Search provides a low-cost alternative, but it requires us to specify the number of random grid locations in advance.

Chapter 5

Conclusions and Future Work

In the following, we will draw some conclusions from the results presented in this thesis and also consider opportunities for future work that could potentially extend these findings.

5.1 Conclusions

In this thesis, we thoroughly investigated the use of Gaussian process learning to optimize the values of parameters in the machine learning domain, where these parameters pertain to a particular base classifier. These tuning parameters are important factors when building a predictive model and thus have substantial impact when predicting target values of new samples with unknown values.

Optimization in the machine learning field is an essential procedure, and every step of the whole data mining process can involve optimization of some form. In order to pursue high performance in terms of different measures, researchers usually attempt many kinds of machine learning algorithms, even ensemble learning to acquire a good model. This can be viewed as optimization. Apart from that, choosing appropriate parameters of any one individual learning algorithm is more likely to generate a good model for a particular dataset,

while unsuitable parameter values easily destroy a model. There are generally some parameters to tune for any particular learning algorithm, a few of which influence very important functions of the model, while the remainder provides little contribution to the predictive results. Hence, we should decide which parameters to optimize before using Gaussian process learning to implement the optimization.

The nature of Gaussian Process Optimization is to combine the characteristics of the Gaussian process algorithm with the rigorous search approach applied in Grid Search. The adaptive optimization methodology that it enables samples new points by maximizing the expected improvement, balancing exploitation and exploration.

This thesis briefly presented an introduction to the optimization context, and described approaches to parameter optimization problems in machine learning. In Chapter 2, we considered various machine learning algorithms with their corresponding parameters to be tuned in Chapter 4. We also considered Gaussian process theory and its mathematical background, and gave the computing process and equations in detail so that the algorithm could be understood clearly. At the end of Chapter 2, we explained the idea of cross-validation, which is applied to evaluate predictive performance.

We focused on Gaussian Process Optimization in this thesis, which was discussed in detail in Chapter 3. Gaussian Process Optimization is able to exploit a Gaussian process predictive model that employs the basic theory of Gaussian process regression to obtain characteristics from collected samples, and thus to predict the target value of a new sample in a user-specified domain. The optimization algorithm searches the surface for likely candidate points, and evaluate them with the expected improvement criterion. We also considered the effects of Gaussian process model parameters on the optimization process, and then considered the stopping criterion that is used to determine when the GPO process stops. Suitable GP parameters for the predictive model used in GPO were decided by experiments based on the "CPU" dataset and the "concrete" dataset.

In Chapter 4, we presented the result of parameter optimization both for regression and classification datasets using the stopping criterion and GP model

parameters that were determined previously. The performance of GPO compared with Grid Search was shown, and we saw that the GPO algorithm achieves a comparably high performance, in conjunction with a smaller number of cross-validation runs and reduced training time in both regression and classification problems.

Another alternative is Random Search, which is a simple but powerful search tool to explore new samples. In this case, the random number generator is used to choose new samples; in other words, there is no relationship between the next chosen point and existing points. Hence, not much time is wasted in this process. Although Random Search does not guarantee that all generated points are points with good performance, the main advantage of Random Search is its low run time. For these reasons, we compared the results of GPO with those of Random Search both in classification and regression tasks. The experiments showed that GPO and Random Search are comparable in most cases in terms of accuracy and correlation coefficient, but in some individual cases, these two algorithms present their own respective advantages. Take decision tree learning (J48) for example: GPO shows statistically significant improvements in accuracy in some cases, such as for the "auto" and "audiology" datasets, no matter how many points are evaluated in Random Search. From the point of view of training time, a dataset with few records or few attributes in each instance consumes less time in Random Search than in GPO when the number of points picked by these two algorithms is approximately same. However, the opposite situation happens for a dataset with larger size. Overall, the GPO algorithm is a promising parameter optimization method in the young area of machine learning.

5.2 Future Work

During the research performed for this thesis, we found that there are many opportunities for future work. We give the following three aspects to improve.

The first question is how to break the limitation in the number of parameters. We only optimized two parameters in this thesis, which potentially prevented some algorithms with multiple parameters from performing ideally.

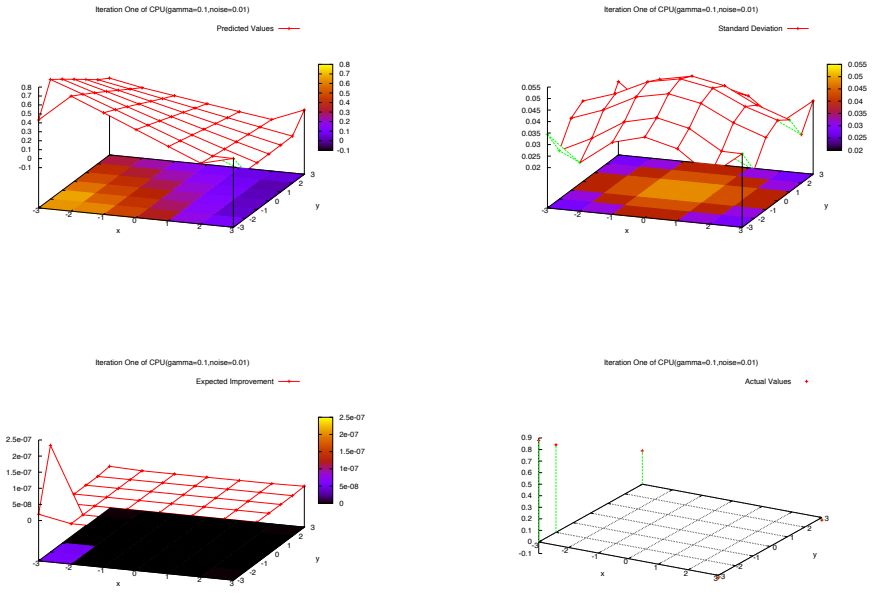
Hopefully in the future, the number of parameters can be determined by the user, and is not fixed in advance. When this is the case, the search space will turn multi-dimension rather than being a grid in two-dimensional space.

The second aspect is that this kind of parameter optimization is used to optimize a known base algorithm. However, in real-world problems, it is often hard to choose an appropriate base algorithm. Practitioners choose algorithms according to the characteristics of the problem and their own experience, or try them one by one. It seems that the optimization process can be viewed as similar to a black-box, which acquires the "best" predictive performance for the given input dataset. Consequently, we can hopefully provide a generalization of the method investigated in this thesis that not only optimizes the parameters of a specific algorithm, but also optimizes the choice of base learning algorithm.

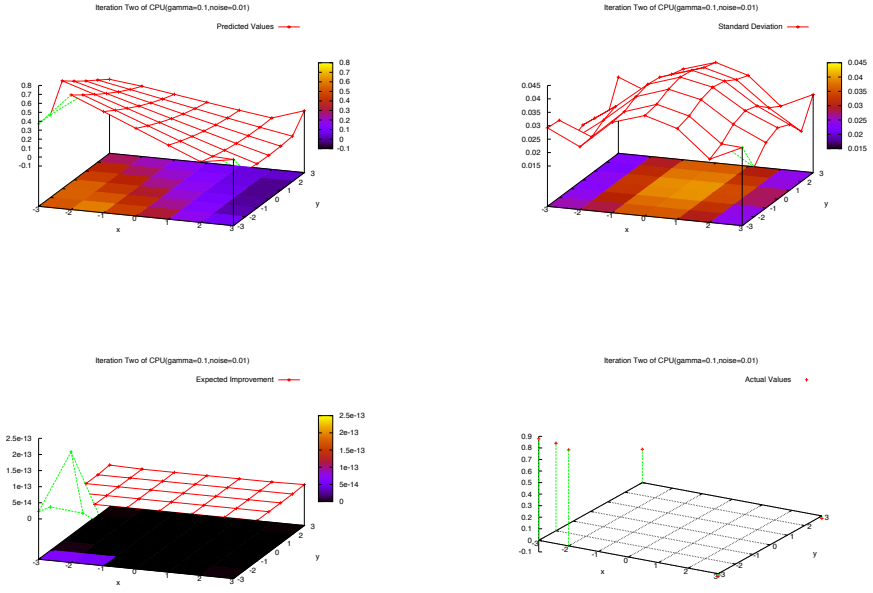
The last limitation of the optimization method considered here is the remaining computational complexity. The optimization algorithm will suffer from this difficulty with increasing size of the datasets involved. A possible way to deal with this problem is to reduce the pressure on one machine and to spread the task to many high-speed computing machines instead, by parallelizing the process.

Appendix A

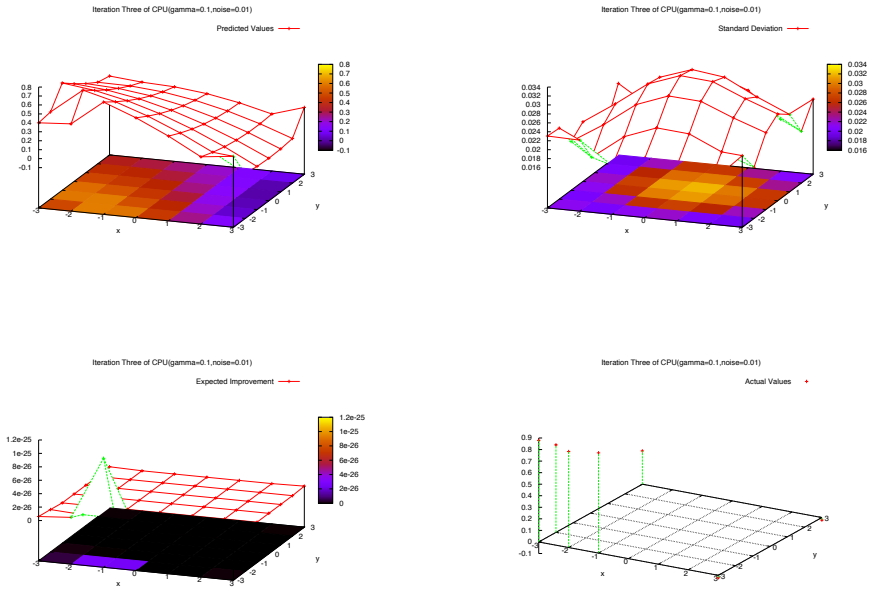
Behaviour on the CPU data with $\gamma = 0.1$ and $\delta = 0.01$ in the first 10 iterations



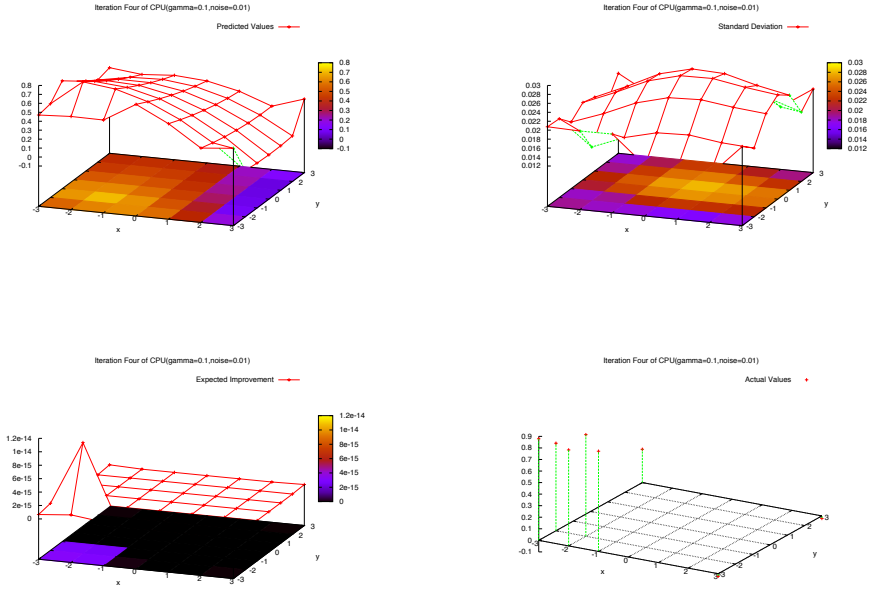
(a) Iteration i



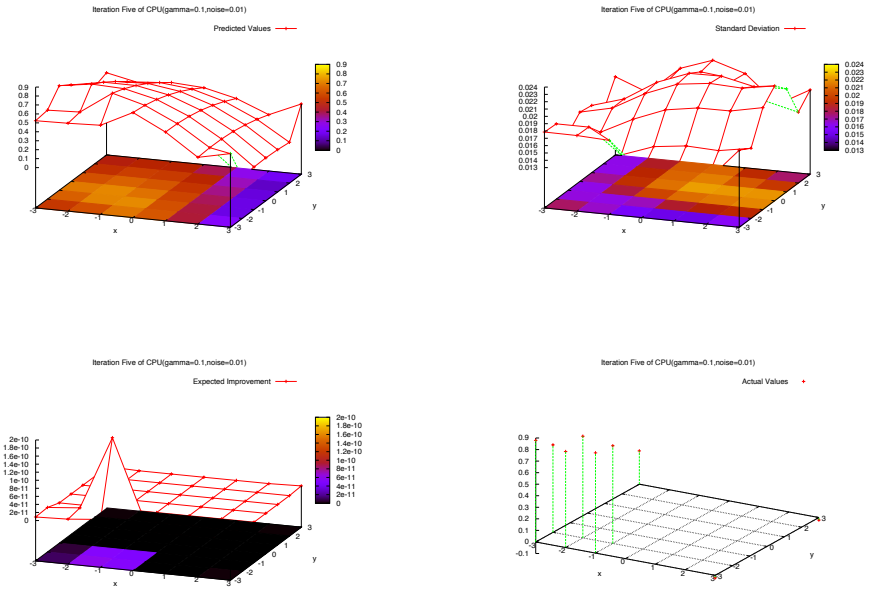
(b) Iteration ii



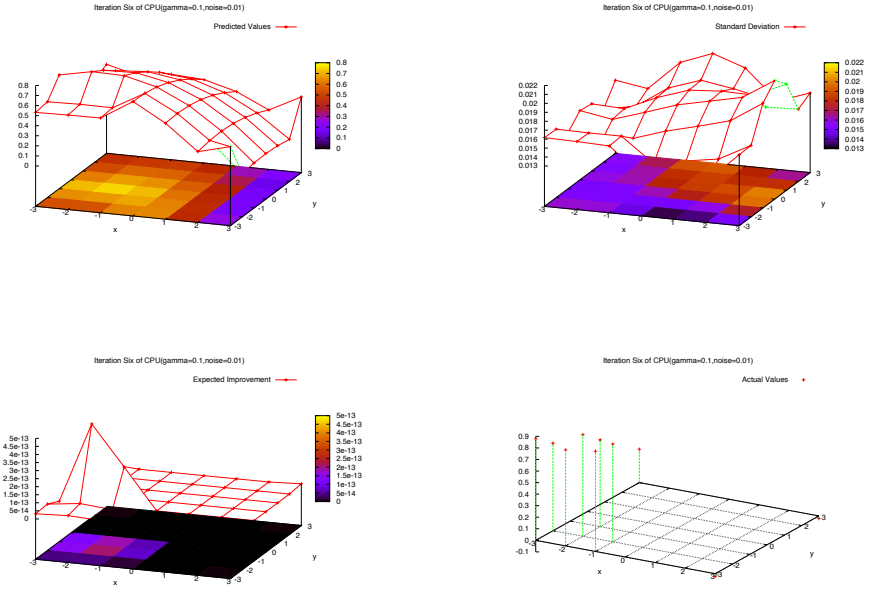
(c) Iteration iii



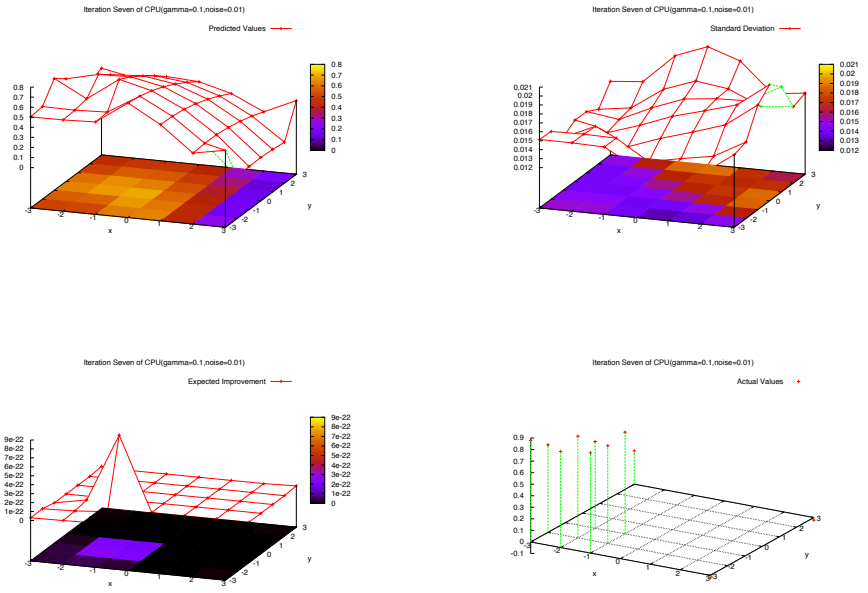
(d) Iteration iv



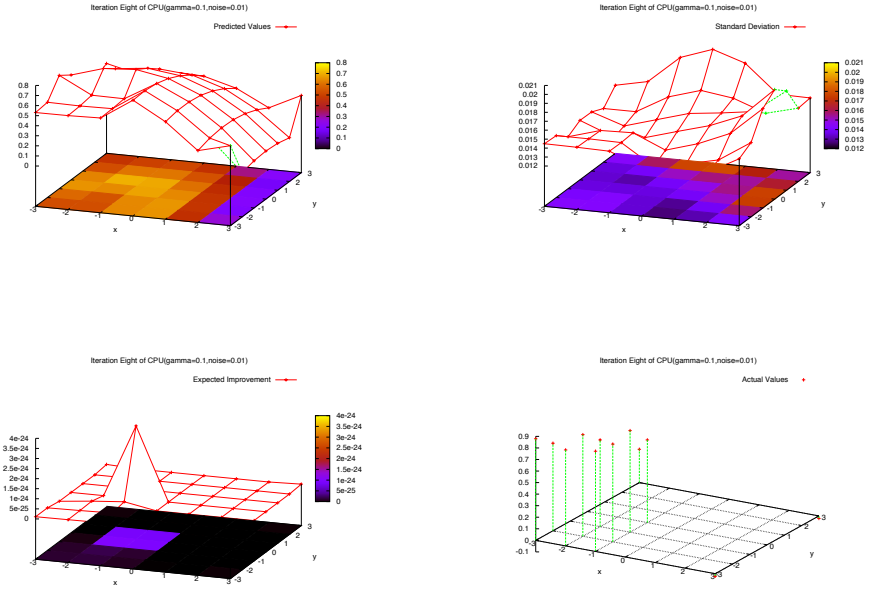
(e) Iteration v



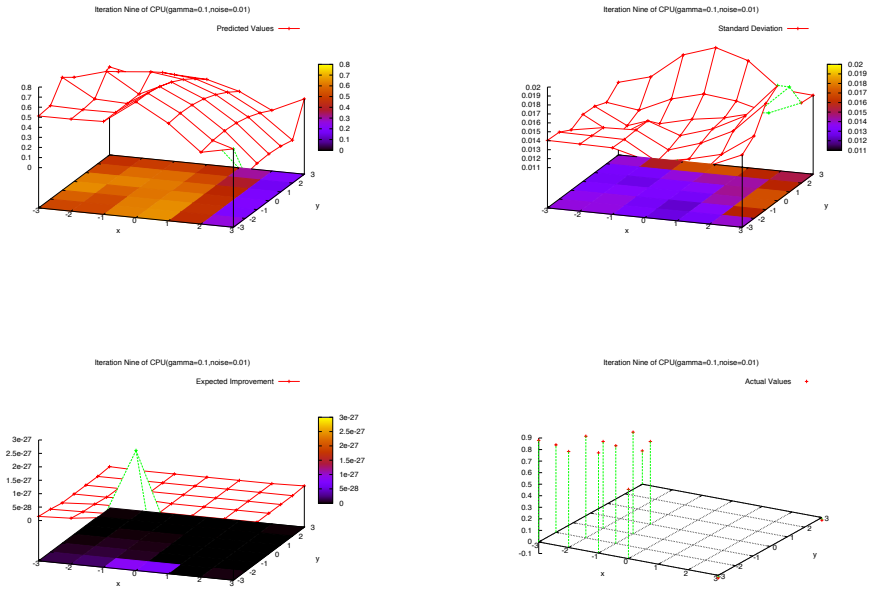
(f) Iteration vi



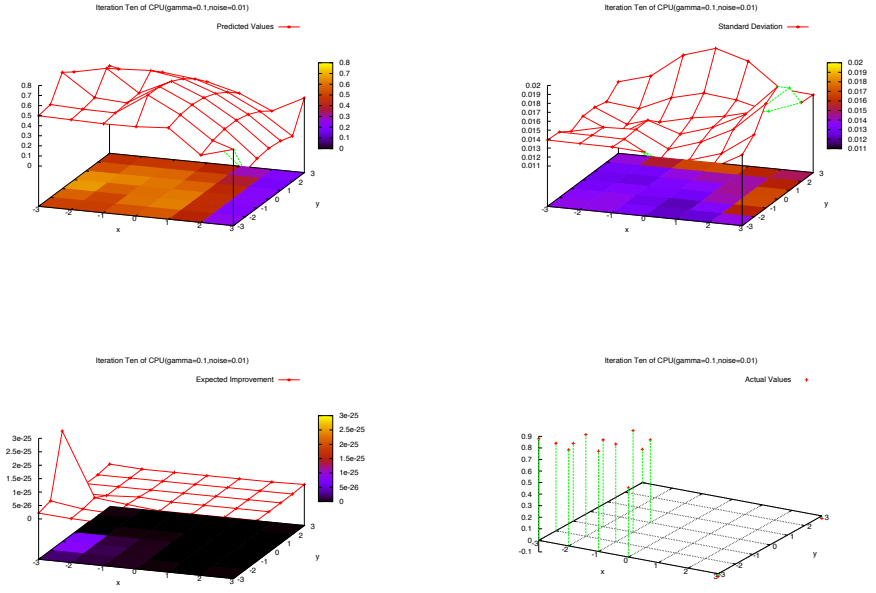
(g) Iteration vii



(h) Iteration viii



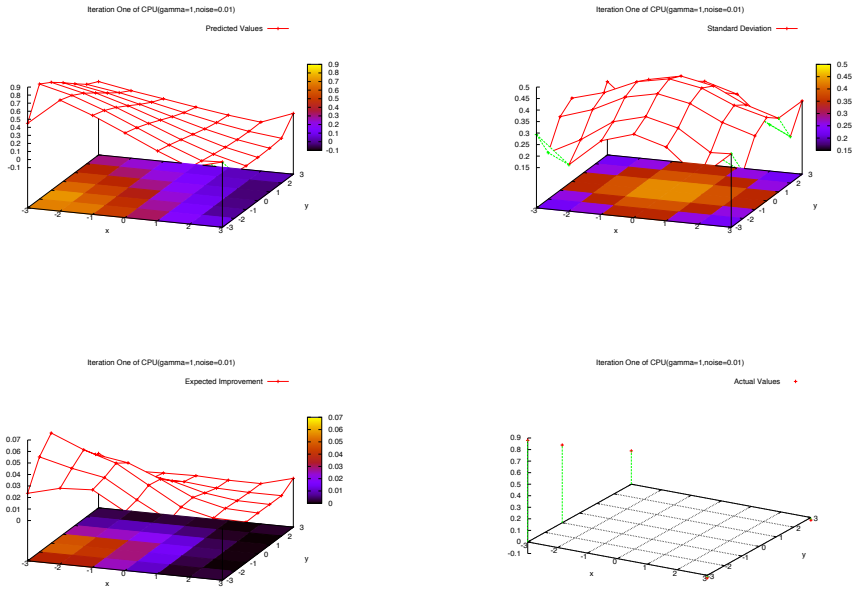
(i) Iteration ix



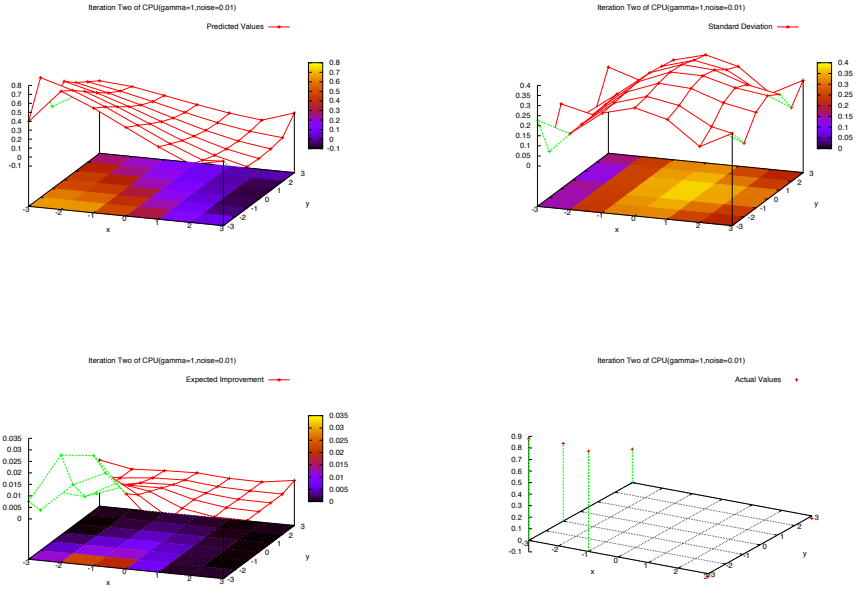
(j) Iteration x

Appendix B

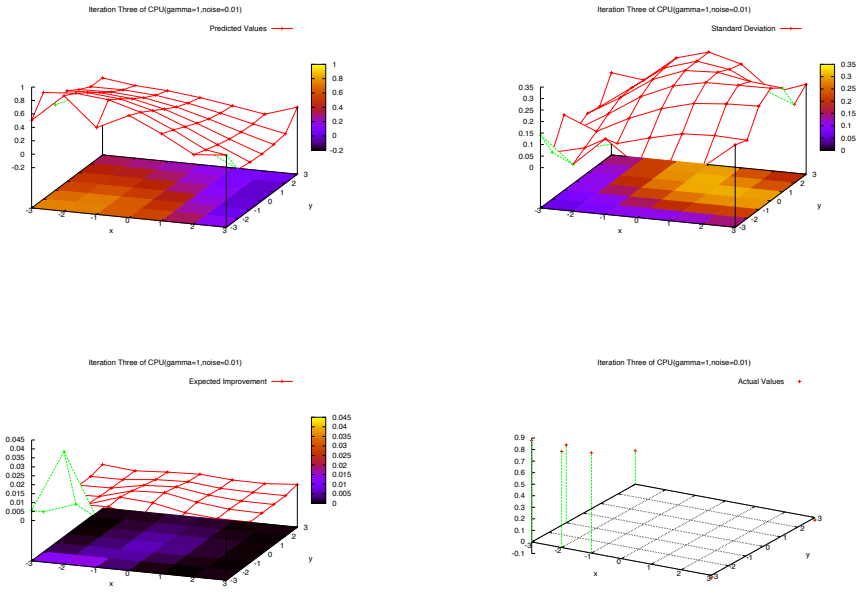
Behaviour on the CPU data with $\gamma = 1$ and $\delta = 0.01$ in the first 10 iterations



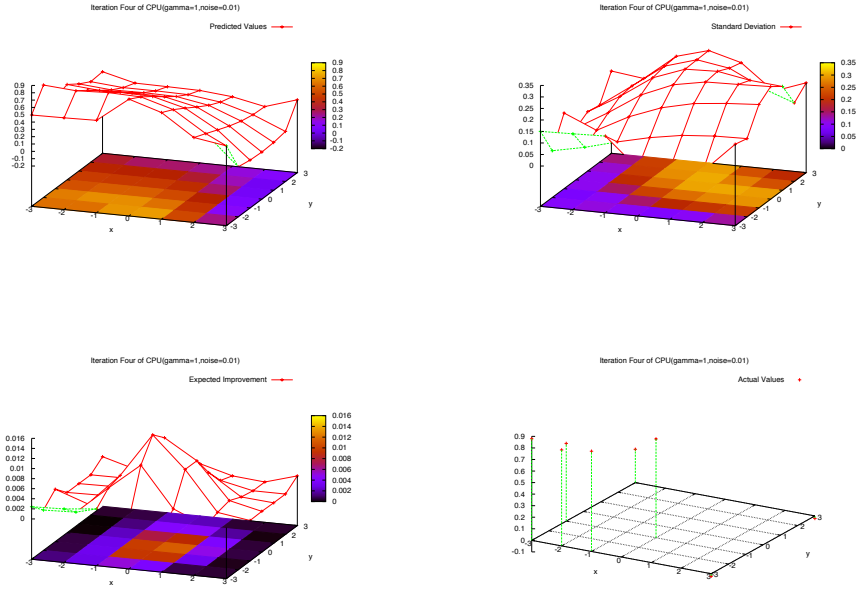
(k) Iteration i



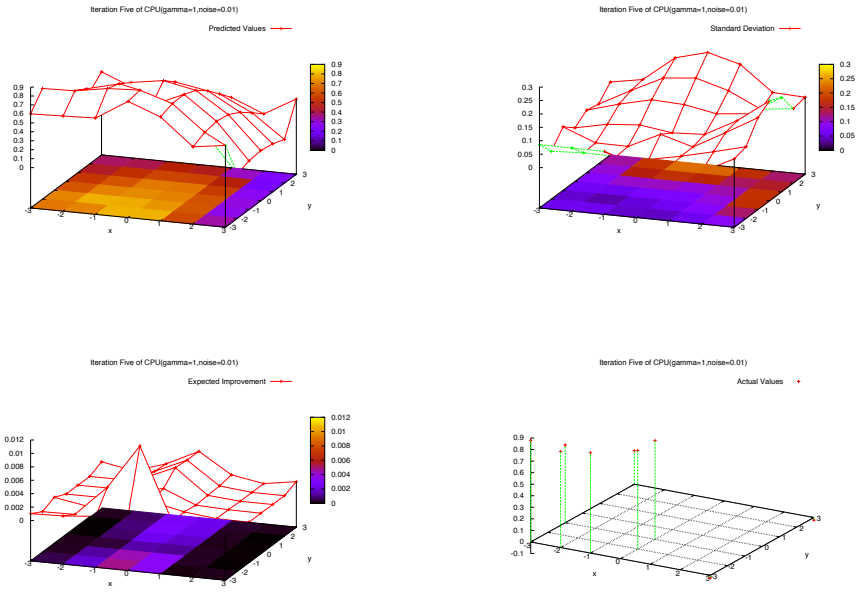
(l) Iteration ii



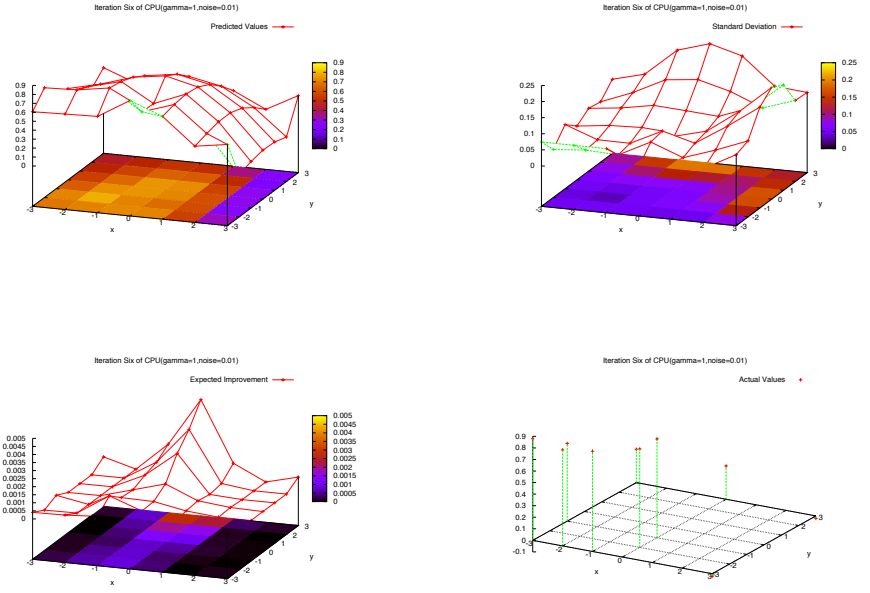
(m) Iteration iii



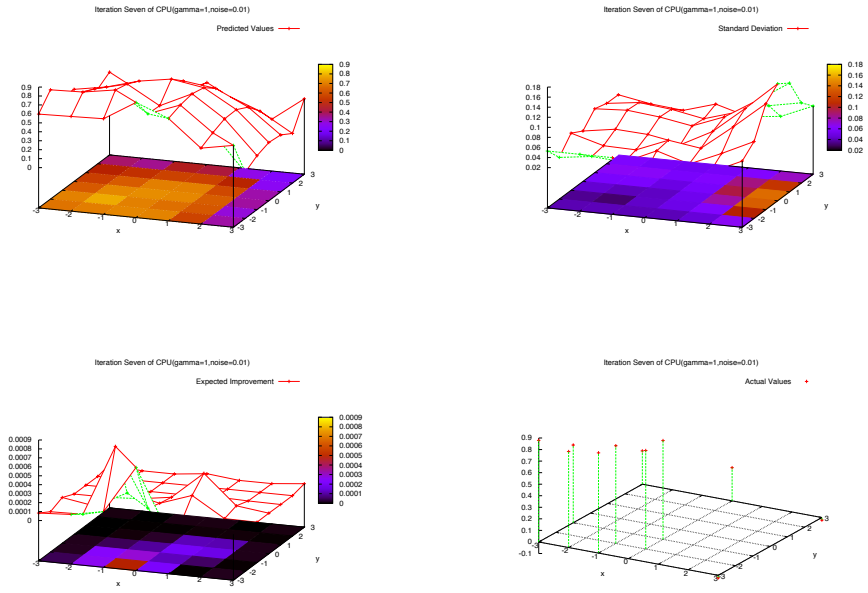
(n) Iteration iv



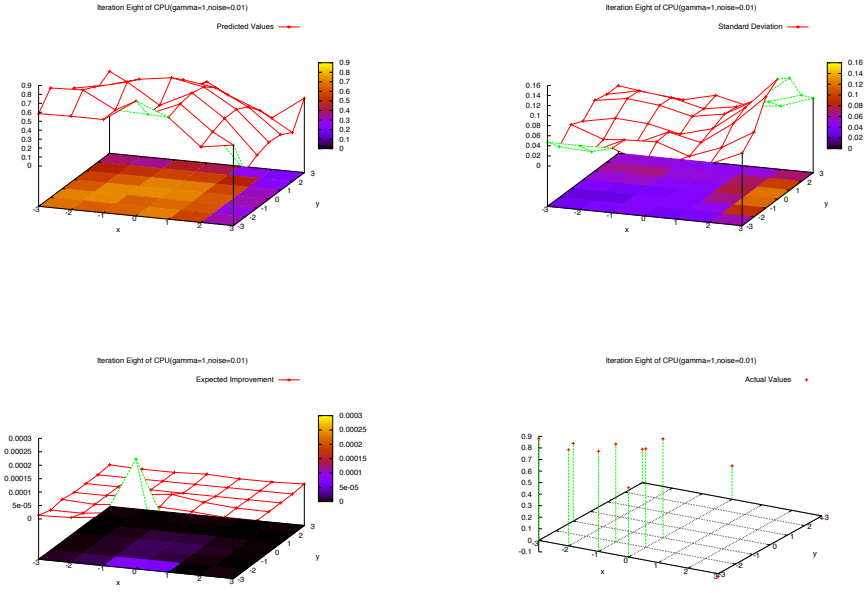
(o) Iteration v



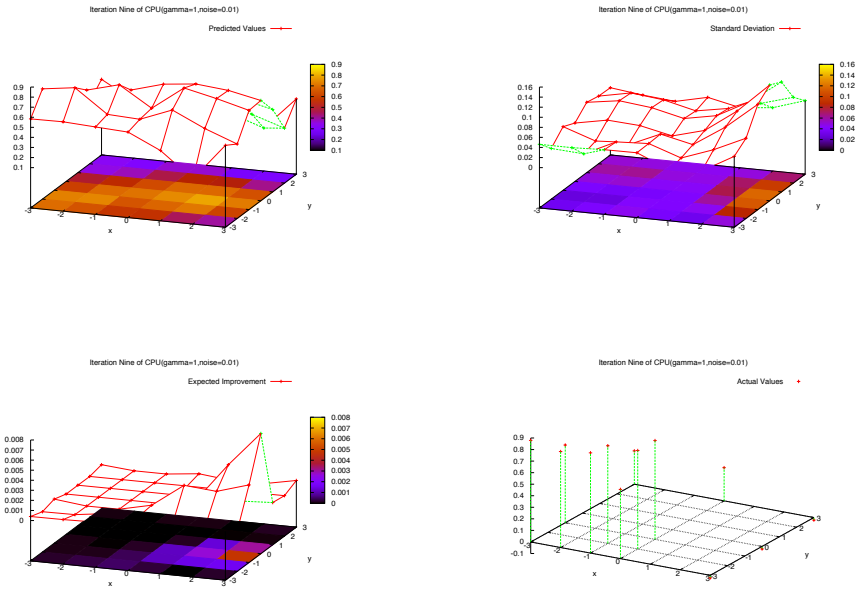
(p) Iteration vi



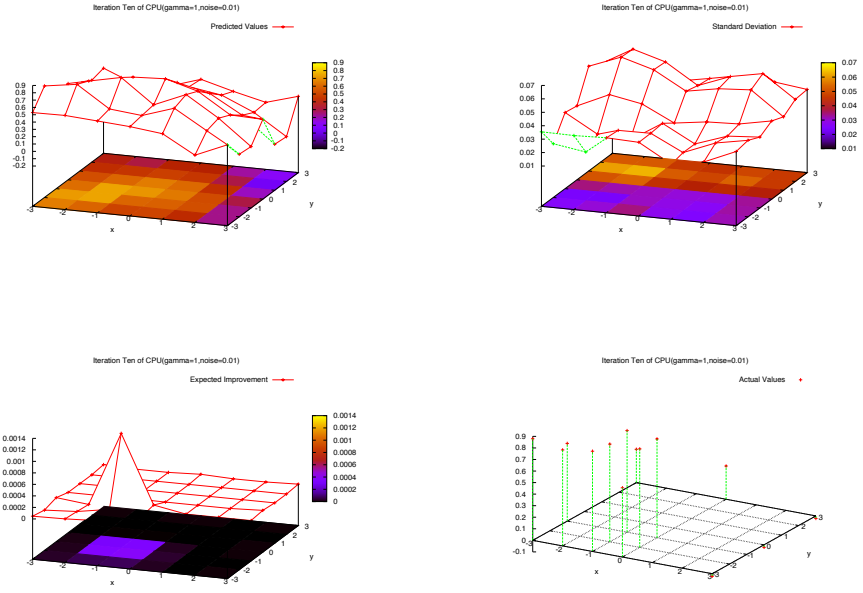
(q) Iteration vii



(r) Iteration viii



(s) Iteration ix



(t) Iteration x

Bibliography

[1] Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k fold cross-validation. *Machine Learning Research*, 5:1089–1105, 2004.

[2] R. Bouckaert, B. Pfahringer G. Holmes, and D. Fletcher. Gaussian processes on graphics cards for NIRS. In *The Fourteenth International Conference on Near Infrared Spectroscopy (NIR2009): Breaking the Dawn*, page 279, 2009.

[3] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[4] C.E.Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[5] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning.*, 20(3):273–297, September 1995.

[6] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[7] H. Drucker, R.E. Schapire, and P. Simard. Boosting performance in neural networks. *IJPRAI*, 7:705–719, 1993.

[8] T. Fawcett. Roc graphs: Notes and practical considerations for researchers. Technical report, HP Laboratories, 2004.

[9] M. R. Frean and P. Boyle. Using Gaussian processes to optimize expensive functions. In *Australian Joint Conference on Artificial Intelligence*, AI '08, pages 258–267. Springer Verlag, 2008.

[10] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121:256–285, 1995.

[11] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings Of The Thirteenth International Conference On Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.

[12] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–407, 2000.

[13] H. Frohlich and A.Zell. Efficient parameter selection for support vector machines in classification and regression via model-based global optimization. In *Proceedings of the IEEE International Joint Conference Neural Networks*, 2005.

[14] T. Onoda G. Rätsch and K.R. Müller. Soft margins for adaboost. *Machine Learning*, 42(3):287–320, March 2001.

[15] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2 edition, January 2006.

[16] T. Hofmann, S. Bernhard, and J. S. Alexander. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, 2008.

[17] G. Holmes, D. Fletcher, and P. Reutemann. Predicting polycyclic aromatic hydrocarbon concentrations in soil and water samples. In *Proceedings of the International Congress on Environmental Modelling and Software (IEMSS), Ottawa, Canada,*, July 2010.

[18] G. Holmes, D. Fletcher, P. Reutemann, and E. Frank. Analysing chromatographic data using data mining to monitor petroleum content in water. In *Information Technologies In Environmental Engineering*, pages 278–290. Springer, May 28-29, 2009.

[19] G.H. John. Cross-validated C4.5: Using error estimation for automatic parameter selection. Technical report, Computer Science Department, 1994.

[20] D.R. Jones, M.Schonlau, and W.J. Welch. Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13:455–492, 1998.

[21] J.R.Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, March 1986.

[22] M. Kearns and L.G.Valiant. Learning boolean formulae or finite automata is as hard as factoring. Technical report, Technical Report TR-14-88, Harvard University Aiken Computation laboratory, August 1988.

[23] M. Kearns and L. G. Valiant. Cryptographic limitations on learning boolean formulate and finite aitomata. *Journal of the Association for Computing Machinery*, 41(1):67–95, January 1994.

[24] M. Kearns and U. Vazirani. *An introduction to Computational Learning Theory*. MIT Press, 1994.

[25] R. Kohavi. *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*, volume 14, pages 1137–1145. Citeseer, 1995.

[26] I. Kononenko. Inductive and Bayesian learning in medical diagnosis. *Applied Artificial Intelligence*, 7(4):317–337, 1993.

[27] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.

[28] D.N. Kumar. Advanced topics in optimization: Direct and indirect. search methods. Technical report, Indian Institution of Science.

[29] L. Licamele and L. Getoor. Predicting protein-protein interactions using relational features. Technical report, UM Computer Science Department, 2007.

[30] S. Lin and S. Chen. Parameter determination and feature selection for C4.5 algorithm using scatter search approach. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 16(1):1–13, 2011.

[31] R.W. Lutz. Logitboost with trees applied to the wcci 2006 performance prediction challenge datasets. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2006.

[32] D.J.C. MacKay. Introduction to Gaussian processes. *Neural Networks and Machine Learning*, 168:133–165, 1998.

[33] C.D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, Cambridge, Mass., 1999.

[34] T. Masaaki and T. Yusuke. Cross-validation, bootstrap, and support vector machines. *Advances in Artificial Neural Systems*, 2011:6, 2011.

[35] A.W. Moore. Entropy and information gain. School of Computer Science, Carnegie Mellon University. http://www.autonlab.org/tutorials/infogain.html.

[36] B. Pant, K. Pant, and K. R. Pardasani. Decision tree classifier for classification of plant and animal micro RNA. *Communications in Computer and Information Science*, 51:443–451, 2009.

[37] J.C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods - Support Vector Learning*, 208:1–21, 1998.

[38] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

[39] R.E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[40] R.E. Schapire. A brief introduction to boosting. In *Proceedings Of The 16th International Joint Conference On Artificial Intelligence*, pages 1401–1406. Morgan Kaufmann., 1999.

[41] R.E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37:297–336, December 1999.

[42] M.J. Shaw and J.A. Gentry. Inductive learning for risk classification. *IEEE Expert: Intelligent Systems and Their Applications*, 5:47–53, February 1990.

[43] C. Staelin. Parameter selection for support vector machines. Technical report, Hewlett-Packard Company, 2003.

[44] A. Sureka and K.V. Indukuri. Using genetic algorithms for parameter optimization in building predictive data mining models. In *Proceedings of the 4th international conference on Advanced Data Mining and Applications*, ADMA '08, pages 260–271. Springer-Verlag, 2008.

[45] M. Umanol, H. Okamoto, I. Hatono, H. Tamura, F. Kawachi, S. Umedzu, and J. Kinoshita. Fuzzy decision trees by fuzzy id3 algorithm and its application to diagnosis systems. In *IEEE World Congress on Computational Intelligence*, 1994.

[46] P. E. Utgoff and C. E.Brodley. An incremental method for finding multivariate splits for decision trees. In *Machine Learning: Proceedings of the Seventh International Conference*. Morgan Kaufmann, 1990.

[47] V.B. Vaghela, A. Ganatra, and A. Thakkar. Boost a weak learner to a strong learner using ensemble system approach. *IEEE International Advance Computing Conference*, 3:1432–1436, 2009.

[48] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[49] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.

[50] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Y Qiang, M. Hiroshi, G.J. McLachlan, A. Ng, B. Liu, P.S. Yu, Z. Zhou, M. Steinbach, D.J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14:1–37, December 2007.