Waseda University Master Thesis

Random Forest Based Quasi-Linear Kernel Composition for Support Vector Machine

44151020-2: ZHAO, Xuan

Master (Engineering)

Professor Jinglu Hu, Supervisor

Neurocomputing System
Information Architecture
The Graduate School of Information, Production and Systems

February 2017

ABSTRACT

Object recognition and classification are one of the most fundamental and representative problems in the artificial intelligence filed. The overcome of such problem will be significant and meaningful. By tons of researches and experiences, it is commonly believed that two factors are essential for the performance of object recognition and classification. They are abstract feature representation and algorithm to optimize a given problem. On the one hand, feature representation, as the only way for a computer to know the real world, determines the upper bound of the potential intelligent level. On the other hand, for a given feature representation and a specific situation, the effectiveness of an algorithm decides the approaching level of the potential upper bound. Therefore, feature representation and algorithm formulation are always two hot topics in the artificial intelligence filed.

In this paper, starting from analysis and summarize the algorithm formulation for classification tasks, local linearity detection is implemented to improve a previously proposed model. Based on it, a local linearity detection based quasi-linear support vector machine is proposed. A quasi-linear here means the nonlinear separation boundary is approximated by using multi-local linear boundaries with interpolation. This thesis proposes a flexible way to construct a quasi-linear kernel by combining the matured ensemble learning method, analysis, and simulation experiments are conducted for exploration of the suitable applications of the proposed method.

Firstly, based on a certain criterion, we train lots of decision trees with resampling data set and random feature subsets. Secondly, we get lots of set of local linear information. Then, after doing cluster, the final information is embedded into the composition of a quasi-linear kernel with the original data set. At last, a global optimization is implemented to train a final classifier.

Experimental results on synthetic datasets show that our proposed method is flexible to capture the potential classification boundary for specific problems, while the experiments in real-world data sets demonstrate improvements of classification performances over traditional classifiers.

Keywords: support vector machine, quasi-linear kernel function, random forest

ACKNOWLEDGMENTS

It is quite an unforgettable time I spend in here, Furuzuki Lab, Waseda University. I, at here, sincerely express my gratitude to my professor, Jinglu Hu, who devoting himself to supervising us. Professor HU is not only knowledgeable but also very willing to help us in daily life. It is a great honor for me to be a member of his Lab.

I would also like to thank Bo Zhou, Weite Li and many other students in Furuzuki Lab who have provided their valuable opinions and assistance to me in my whole master period.

At last, thanks to all my friends and my parents for their great support and companion, this paper can not be finished without their patient to comfort me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, a flexible way based on ensemble learning method to compose a quasi-linear kernel for support vector machine is proposed for a better performance of binary classification tasks. In this chapter, firstly, we will introduce the background of machine learning field and it's significant. Next, we will introduce the related works of this thesis. Then, the motivations and the purpose of this method will be presented. Last, the structure of this thesis will be laid out to give a perspective view.

1.1 Machine Learning for Classification

As a way of realizing artificial intelligence, machine learning has been widely accepted and practiced with the help of the modern computer. As an important subject in the field of machine learning, the problem of classification has been paid high attention to the experts in the field since the birth of machine learning.

Classification problem is defined as that building a model under given data distribution, then making the computer can automatically classify input data and give it a class tag [1]. For example, the image stored in computer memory just a bunch of binary numbers which humans can not understand. However, the computer could extract some clear features of the characteristics from the given original binary numbers by learning algorithm, and the relevance of these features is the key components to rebuild a specific image. So, once we have the feature extraction method and tagged label information for each image, and then split these image data set into the training set and test set. Then, the computer can use the training set for classifier learning, using the test set to test the performance of the trained classifier. Therefore, if we can design an effective learning algorithm, the computer will be able to self-learning, get the capability of automatic classification of different images [2].

Generally speaking, classification problems can be categorized into supervised learning and unsupervised learning. In a supervised setting, a corresponding learning process is conducted with the help of not only input patterns but also their labels. However, in an unsupervised setting, only input patterns are provided for the learning process. By a simple comparison, the learning processes of supervised learning contain much more information of samples, so attract major attentions from researchers. In this thesis, supervised learning is our major concern.

There are so many discriminative supervised learning technologies in the machine learning field, such as logistic regression [3], support vector machines [4, 5], neural networks [6] and so on. Unlike other methods, SVM uses structural risk minimization criteria to minimize the upper bound of the generalization error rather than simply reducing the model's training error. Moreover, the optimization process of an SVM is a convex optimization process, so it can guarantee a global minimum solution. Besides, SVM introduces a kernel trick [7], which could map the original feature space to a very high dimension feature space. By ensuring the linear separability of the input samples in the high dimensional space, the SVM is promoted from a powerful linear classifier to a more powerful nonlinear classifier. For these reasons, SVM has received extensive attention in industry and academia for its excellent performance in classification.

However, it is worth noting that since the traditional kernel functions (such as radial basis function) construct only a black box mapping for spatial variation, it does not take into account the structural information of the input data itself. Therefore, in the traditional mapping of kernel functions, the actual samples, and the noise samples will be simultaneously mapped to linearly separable high-dimensional space. As a result, the traditional SVM will inevitably have the problem of over-fitting, that is, the final classification boundary will over-fit the noise distribution and can not match the potential structure of the actual samples distribution itself. In the situation of high noise data or the number of samples is close to or much larger than the number of samples, the over-fitting problem is particularly severe. [8, 9].

Inspired by the use of piecewise linear classifier approximating nonlinear boundary hyperplane, SVM based on quasi-linear kernel function is used to solve the over-fitting problem [10, 11]. By using the local linear information extracted from the feature space of the samples in the kernel function, the constructed kernel function can express the mapping method of the original input sample, and can also describe the actual physical meaning in the mapping process.

After searching a lot of literature and exploring related methods, it is concluded that the robustness of local linear information directly determines the final training effect of quasi-linear kernel support vector machine. In view of this, this thesis proposes an ensemble learning method based on decision forest. The local linear information of the input samples learned by this method is taken as the core component of the composition of the quasi-linear kernel function.

1.2 Related Methods

In order to solve the problem of over-fitting of kernel-based SVM in training process, the idea of constructing a combinational linear hyperplane to approximate non-linear hyperplane by using multiple SVMs is explored. But this type of SVM still could not guarantee the solution is globally optimal.

In the literature [12], J. Hu et al. (2001) proposed a quasi-linear ARX model to identify a nonlinear system. The quasi-linear ARX model can be regarded as a nonlinear approximation model with multiple interpolated linear models and has good generalization ability in the identification of nonlinear systems [13]. The model was latter extended to a quasi-linear support vector machine for solving classification problem [13, 12]. By this way, they transform the PLC models from multiple local linear models optimization process to kernel function optimization so that the final nonlinear classifier is just one SVM. Because we are aiming at a support vector machine optimization process, we can guarantee that the solution is a globally optimal solution.

Therefore, in this thesis, we focus on this kernel way to construct a multi-local linear classifier, namely quasi-linear kernel SVM. Firstly, let's talk about the relative branch of SVMs research briefly, namely, using piecewise linear classifier approximating nonlinear boundary hyperplane.

1.2.1 Unsupervised learning method

Unsupervised learning method which doesn't use label information, such as Kmeans, KNN, Manifold Learning methods. H. Zhang proposed a method which using K-nearest neighbor algorithm, for each sample in the test set, find the adjacent K samples in the train set to construct a local linear classifier [14] . By this method, the SVM is transformed from global optimization to local optimization to reduce the complexity of the model and improve the performance of the classifier by using local linear separable properties. H. Cheng proposed a method which is to construct multiple linear classifiers in the training data set and each linear classifier corresponds to a subspace in the test samples feature space and ensure that the positive and negative samples in these subspaces can be separable by a linear hyperplane [15]. Since multiple linear SVM have the advantages that they are locally robust, for some noisy nonlinear classification problems, this type of methods may perform better than general nonlinear kernel SVMs [16].

1.2.2 Semi-supervised learning method

In semi-supervised learning method, B.Zhou proposed a modified k-means method which also considers the information of label and proportion of label [11]. Firstly, by introducing the label information of the samples into the K-means algorithm, the division of the sample feature space is obtained, and then all the training samples are clustered. The local linear partitions information are used to construct the quasi-linear kernel function, and the global optimization can be ensured as well as the traditional support vector machine (SVM).

1.2.3 Supervised learning method

In this kind of work, a supervised method would be more suitable for the determination of local partitions (we will discuss it in the next chapter). Therefore, it is highly motivated to develop a more sophisticated method to detect them properly. In the previous work, the geometric method could achieve better results [17].

1.3 Motivation

So far, we already know the kernel way to construct a multiple local linear classifier could have better performance than other ways in theoretic reason. However, two important issues that need to be addressed for local linear classifiers.

Remark 1 *How to detect the potential local linearity information.*

Remark 2 *It is still difficult to determine the number of local linear classifiers and their respective scopes.*

The major reason behind it is that a data-independent kernel function is hard to represent the complicated manifold where samples concentrate [18]. By this way, since we must construct a quasi-linear kernel function, we must first find a way to detect the local linearity and then, we also must know the number of local linear classifiers and their respective scopes. For this reason, *the proposed method in this paper would give a more flexible and robust supervised algorithm for composition a quasi-linear kernel.*

It is expected that the information of local linearity can be firstly detected by our proposed method, which means that the number of local linear classifiers and their function scopes should be firstly determined by our method. After this, the information can be embedded into the composition of the quasi-linear kernel. Then, a global optimization can be expected to output a compelling result.

1.4 Thesis Organization

The remainder of this article is organized as follows: In Chapter 2, introduce the concepts, definitions and provide mathematical solutions of quasi-linear kernel SVM, and discuss the inner mechanism of the quasi-linear SVM. In chapter 3, several classical algorithms of a decision tree theory are introduced. Then show decision tree can be used to get local linear information. Next, we introduce random forest, which is a powerful extension to the decision tree algorithm in statistical machine learning aims to get a preciser local linear information for kernel construction. In Chapter 4, the feasibility and effectiveness of the proposed method are demonstrated by simulating experiments. Finally, conclusions are discussed in Chapter 5.

Chapter 2

Quasi-linear Support Vector Machines

In this chapter, we will first introduce the necessary contents of SVM, then introduce the related technology of quasi-linear vector machine, analyzes its working principle, and discusses the optimization approach of the quasi-linear kernel function.

2.1 Support Vector Machine

SVM as a supervised learning method [5], which is widely used in classification and regression problems. Because at that time its performance even surpassed the complex neural network system, therefore has caused the great attention and the research interest in the academic circle. Because it uses the optimization learning method developed from the statistical learning theory, it learns a linear classification function by using structural risk minimization criterion in the high-dimensional sample space. Compared with the empirical risk of traditional neural networks, the structural risk minimization criterion directly optimizes the upper limit of the expected risk, which shows the learning and generalization performance better than the risk minimization criterion.

2.1.1 Support Vector Machine Optimization

We start from basic case, given number of L training samples, each sample contains a d dimensional input signal $(x_i \in \mathbf{R}^d)$ and a class label $(y_i \in -1, 1)$. First, the hyperplane of all $\mathbf{R}^d$ spaces is represented by a vector ω and a constant b as the following equation:

$$\omega^T x + b = 0 \tag{2.1}$$

where ω is a vector orthogonal to the hyperplane and b is the bias of the hyperplane. Assuming a hyperplane to divide the data set, then have the following equation:

$$f(x) = sign(\omega^T x + b) \tag{2.2}$$

In the case of linear separability, the hyperplane is defined as having a margin of at least unit one from the data set, namely the hyperplane satisfies the following condition:

$$\omega^T x_i + b \geq +1 \quad \text{when } y_i = +1 \tag{2.3}$$

$$\omega^T x_i + b \leq -1 \quad \text{when } y_i = -1 \tag{2.4}$$

Or just write as:

$$y_i \left(\omega^T x_i + b\right) \geq 1 \text{ for } \forall i \tag{2.5}$$

In fact, (ω, b) and $(\lambda\omega, \lambda b)$ where $\lambda \in \mathbf{R}^+$ represent the same hyperplane, since each hyperplane has a different functional distance for a given training data set. Therefore, in order to get the distance between the hyperplane and the data set, we must normalize the length of vector w, that is, rewrite above formula as:

$$d((\omega, b), x_i) = \frac{(\omega^T x_i + b)}{||\omega||} \geq \frac{1}{||w||} \tag{2.6}$$

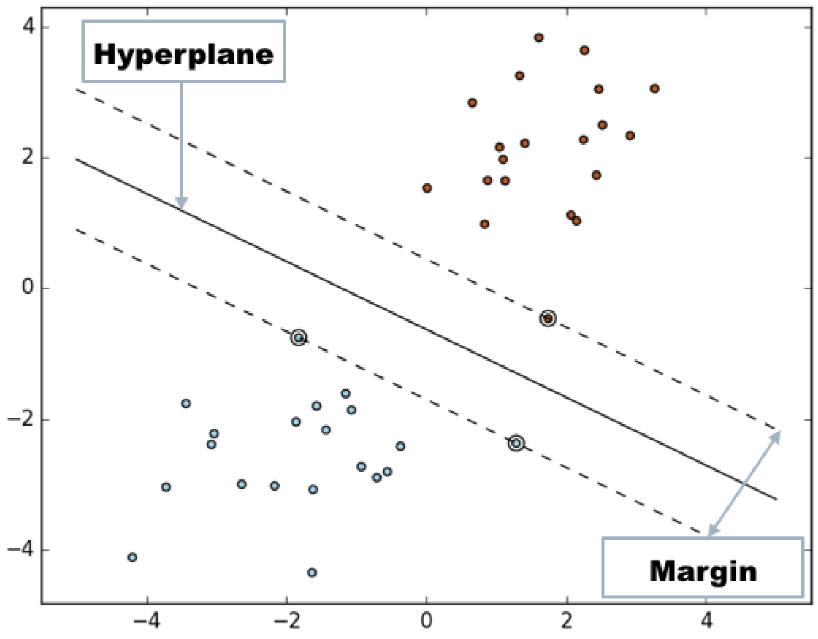


Figure 2.1 Illustration of a hyperplane and maximum margin for a SVM

Thus, the optimal hyperplane can be defined as maximizing the margin between the hyperplane and the data set, as shown in the following figure 2.1, circled points are called support vectors.

At this point, the support vector machine hyperplane optimization problem can be converted to maximum $\frac{1}{||w||}$under the constraints. In summary, the support vector machine optimization formulation can be written as follows:

$$\begin{aligned} &\min_{\omega} \frac{1}{2}\|\omega\|^2 \\ &s.t.\ y_i\left(\omega^T x_i + b\right) \geq 1,\ i = 1,2,...,m \end{aligned} \tag{2.7}$$

This formula is a standard convex quadratic programming problem, although the results can be obtained directly by software, but if the input signal dimension is too large, will lead to the optimization process slowing down, and even lead to unenforceable. In order to solve such problem, the dual form of this formula is widely used.

According to Lagrangian duality, Lagrange Multiplier $\alpha_i, i = 1,2,\ldots,m$ is introduced, formula 2.7 could rewrite as:

$$L(\omega,\alpha,b)=\frac{1}{2}\|\omega\|^2-\sum_{i=1}^{m}\alpha_i\left[y_i(\omega^T x_i+b)-1\right] \tag{2.8}$$

In order to derive the dual form of the equation, first need to fix α_i, then calculate the partial derivatives of ω, b to optimize $L(\omega,\alpha,\beta)$:

$$\nabla_w L(\omega,b,\alpha)=\omega-\sum_{i=1}^{m}\alpha_i y_i x_i=0 \tag{2.9}$$

$$\frac{\partial}{\partial b}L(\omega,b,\alpha)=\sum_{i=1}^{m}\alpha_i y_i=0 \tag{2.10}$$

According to Karush-Kuhn-Tucker conditions (KKT), we could know the primal and dual problem are the same solutions in SVM. So, we could get the dual optimization formulation of support vector machine finally:

$$\begin{aligned}&\max_{\alpha}\ W(\alpha)=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}y_i y_j\alpha_i\alpha_j<x_i,x_j>\\ &s.t.\ \alpha_i\geq 0,\ i=1,2,...,m\\ &\quad\sum_{i=1}^{m}\alpha_i y_i=0\end{aligned} \tag{2.11}$$

After solving parameter of α_i, then we can solve ω from formula 2.11, finally get b from the formula below:

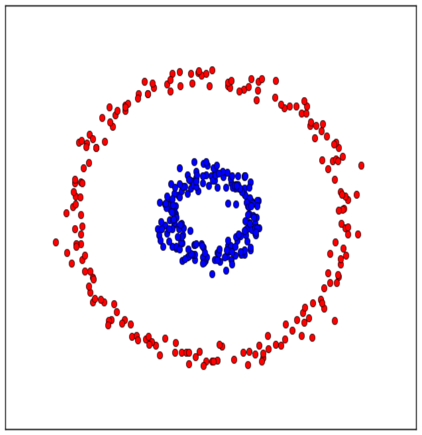
$$b^*=-\frac{\max_{y_i=-1}\omega^{*T}x_i+\min_{y_i=1}\omega^{*T}x_i}{2} \tag{2.12}$$

Finally, using formula 2.2 calculate $f(x)$:

$$\begin{aligned}f(x)&=sign(\omega^T x+b)=sign(\sum_{i=1}^{m}\alpha_i y_i x_i)^T x+b\\ &=\sum_{i=1}^{m}\alpha_i y_i<x_i,x>+b\end{aligned} \tag{2.13}$$

Therefore, only use α_i, we can obtain final classification result by calculating the inner product between the test sample and train samples. Notice that only support vectors have α_i larger than 0, so-called support vector machine.

2.2 Kernel Function


(a) A nonlinear separation data set

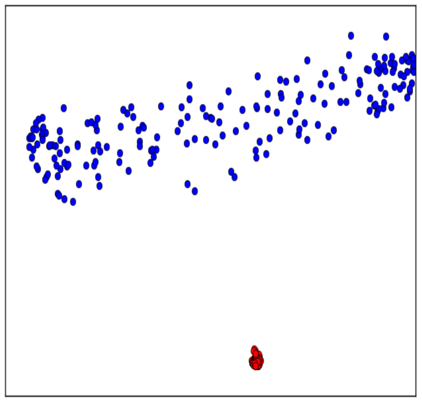

(b) An illustration of KPCA feature transformation

Figure 2.2 Visual kernel mapping

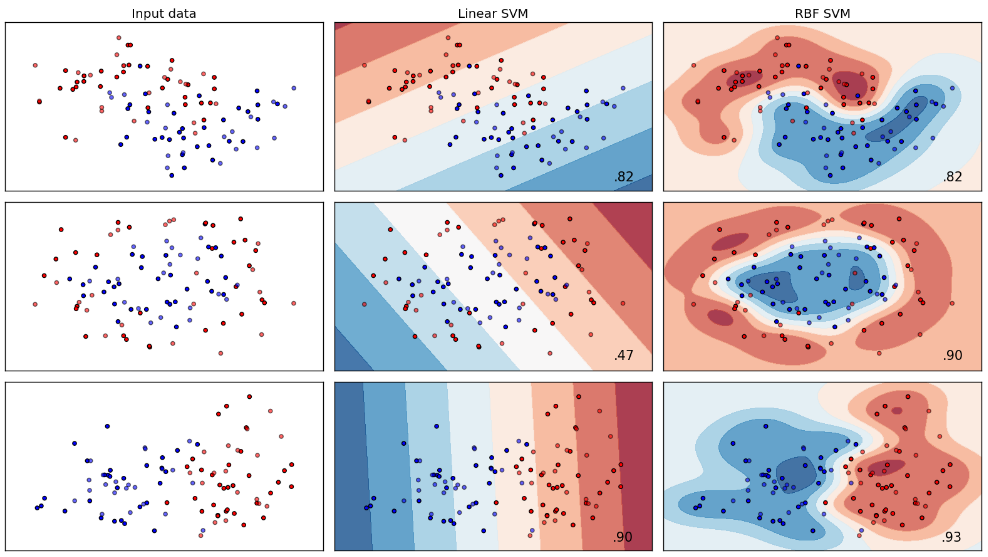


Figure 2.3 A comparison of linear SVM and nonlinear SVM on synthetic datasets.

For non-linear separable data sets in figure 2.2(a), SVM implicitly map the given data set to a linearly separable high dimensional space by using a kernel function $\mathbf{K} < \cdot, \cdot >$, visualization of the KPCA projection figure 2.2(b) [19]. The kernel SVM be up-grate to a non-linear classifier could be seen in figure 2.3, training points in solid colors and testing points semi-transparent, the lower right shows the classification accuracy on the test set. Since the kernel function is used as an extension of the inner product of the data points, and the size of the product matrix is only related to the data size. The nonlinear expansion obtained by using the kernel function, in the case of small data size, the consume in memory and computation do not result in more complexity. This excellent property makes the application of kernel function very extensive. This section gives a definition of kernel functions and a brief description of the application of kernel functions in support vector machines. Finally, we will discuss how to determine whether a function is a kernel function without constructing a mapping function, which will play a crucial role in deriving quasi-linear kernel functions in later chapters.

2.2.1 Definition of Kernel Function

Let X be the input space (a subset of the European space $\mathbf{R}^n$ or a discrete set), and H be the feature space to which the data is projected from the input space through the mapping operator [7]. For all $x, z \in \mathrm{X}$, if it exits a mapping from X to H,

$$\phi(x) : \mathrm{X} \rightarrow \mathrm{H} \tag{2.14}$$

to ensure that the function $\mathrm{K} < x, z >$ meet the conditions,

$$\mathrm{K} < x, z >= \phi(x)^T \phi(z) \tag{2.15}$$

then called $\mathrm{K} < x, z >$ for the kernel function, $\phi(x)$ for the mapping function.

It can be seen from the definition of the kernel function that the mapping function $\phi(x)$ is usually not explicitly defined in the way of using the kernel function, but directly solves the problem by defining and constructing the kernel function K $< x,z >$. Review the dual problem of the linear support vector machine deduced in the previous section formula 2.11. In its optimization goal, only involves the inner product operation $< x_i,x_j >$, therefore it can be replaced by the kernel function K $< x_i,x_j >$, rewrite the formula 2.11 as:

$$\begin{aligned} \max_{\alpha}\ W(\alpha) &= \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K < x_i,x_j > \\ s.t.\ \alpha_i &\geq 0,\ i = 1,2,...,m \\ \sum_{i=1}^{m} \alpha_i y_i &= 0 \end{aligned} \tag{2.16}$$

In the same way, the inner product $< x_i,x_j >$ in the classification decision function 2.13 can be replaced by the kernel function, thus we can get:

$$\begin{aligned} f(x) &= sign\Big(\sum_{i=1}^{m} \alpha_i y_i \phi(x_i)\Big)^T \phi(x) + b \\ &= \sum_{i=1}^{m} \alpha_i y_i K < x_i,x > +b \end{aligned} \tag{2.17}$$

The derivation from the mathematical level explains the kernel function can be used to solve the nonlinear support vector machine approach by solving the solution of linear support vector machine, which is mentioned in the beginning of this section. This technique is often referred to as kernel trick. In actual cases, the choice of the kernel function will greatly affect the final performance of the classifier.

2.2.2 Positive Definite Kernel Function

Since it is necessary to ensure that the designed kernel function is a positive definite kernel function from the viewpoint of theoretical derivation, the sufficient and necessary conditions for positive definite kernel function will be given. The results will provide theoretical evidence for the proof of the quasi-linear kernel function defined in the following section.

Assuming that the kernel function K $< x,x >$: X $\times$ X $\rightarrow$ R is a symmetric function, then K $< x,z >$ is a positive definite kernel function if and only if the Gram matrix K $= [\text{K} < x,z >]_{m\times m}$

corresponding to the kernel function $\mathrm{K} < x, z >$ is a positive definite matrix for any $x_i \in \mathrm{X}, i = 1, 2, \ldots, m$.

2.3 Quasi-Linear Support Vector Machine

In this section, we introduce quasi-linear kernel support vector machine theory. In the quasi-linear vector machine model, multiple sets of local linear models are combined with corresponding local threshold functions to form a nonlinear boundary hyperplane [11, 13]. Different from the conventional method, the quasi-linear support vector machine can obtain the global nonlinear model by integrating all the local models in the kernel space so that the global optimal solution can be obtained only by one global optimization process. The constructed kernel function is called quasi-linear kernel function, which is the key to optimizing the quasi-linear support vector machine by the traditional kernel support vector machine optimization model. In order to construct the quasi-linear kernel function, the common method is to split the local feature space and extract the local information by using the label information to obtain the local threshold function parameters which are needed to construct the quasi-linear kernel function.

2.3.1 Mathematical Model of Quasi-Linear Support Vector Machine

In the case of binary classification problems, suppose we have a training set containing L labeled data points $\{x_i, y_i\}, i = 1, 2, ..., l$, and $x_i \in \mathbf{R}^d$ with a label $y_i \in \{-1, 1\}$. First, considering a binary classifier $f(x)$ can be described as:

$$f(x) = g(x), \quad x \in \mathbf{R}^d \tag{2.18}$$

where $g(x)$ is a nonlinear function. Applying Taylor expansion to $g(x)$ around the region $x = 0$, we can get:

$$f(x) = g(0) + \left(g'(0) + \frac{1}{2} x^T g''(0) + ... \right) x \tag{2.19}$$

and by defining a vector as:

$$\theta(x) = \left(g'(0) + \frac{1}{2}x^T g''(0) + \dots\right)^T \tag{2.20}$$

then we get a regression form of the classifier

$$f(x) = g(0) + x^T \theta(x) \tag{2.21}$$

where $\theta(x)$ is a vector of unknown functions of x. By introducing an basis function based networks to parameterize the unknown vector $\theta(x)$ [20], we get

$$\theta(x) = \sum_{j=1}^{M} \Omega_j R_j(x) + \Omega_0 \tag{2.22}$$

$$g(0) + x^T \Omega_0 = \sum_{j=1}^{M} b_j R_j(x) + b \tag{2.23}$$

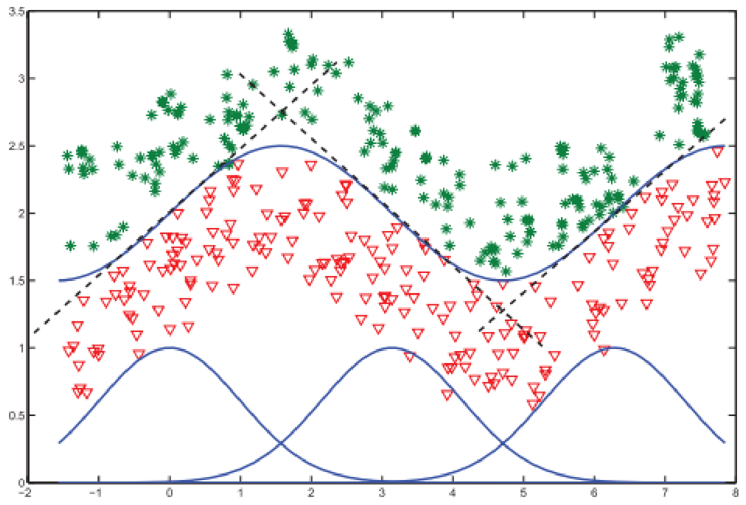


Figure 2.4 Multiple local linear classifiers with interpolation for nonlinear separation hyperplane.

As a result, we can regard the binary classifier $f(x)$ is constructed by combining with number of M local linear models with local threshold function (see figure 2.4)[11]. It can be further defined as follows:

$$f(x) = \sum_{j=1}^{M} \left(\Omega_j^T x + b_j\right) R_j(x) + b \tag{2.24}$$

where $\Omega_j (j = 1, 2, ..., M)$ and $b_j (j = 1, 2, ..., M)$ is the parameters of j-th local linear classifier, b is the constant parameter and $R_j(x)(j = 1, ..., M)$ is the local threshold basis functions corresponding each local linear model, it satisfies that $\sum_{j=1}^{M} R_j(x) = 1$.

Now, we introduce two vectors $\Phi(x)$ and $\Theta(x)$ defined by

$$\Phi(x) = \left[R_1(x), x^T R_1(x), ..., R_M(x), x^T R_M(x)\right] \tag{2.25}$$

$$\Theta = \left[b_1, \Omega_1^T, ..., b_M, \Omega_M^T\right] \tag{2.26}$$

the Eq.(2.22) can be further expressed as:

$$f(x) = \Theta^T \Phi(x) + b \tag{2.27}$$

By applying the structural risk minimization principle to the Eq.2.27, then a binary classification problem solving process could be described as a QP optimization problem:

$$\begin{aligned}
&\min_{\Theta, b, \xi_k} J_p = \frac{1}{2}\Theta^T \Theta + C \sum_{k=1}^{N} \xi_k \\
&s.t. y_k \left[\Theta^T \Phi(x_k) + b\right] \geq 1 - \xi_k, \forall k \\
&\xi_k \geq 0, \forall k
\end{aligned} \tag{2.28}$$

Then, by introducing new Lagrange multipliers (α_k, ν_k), Lagrange function can be constructed in the following forms:

$$\begin{aligned}
&L(\Theta, b, \xi; \alpha, \nu) = J_p(\Theta, \xi) \\
&- \sum_{k=1}^{n} \left(\alpha_k y_k \left[\Theta^T \Phi(x) + b\right] - 1 + \xi_k\right) - \sum_{k=1}^{N} \nu_k \xi_k
\end{aligned} \tag{2.29}$$

Where $\alpha_k \geq 0, \nu_k \geq 0$. The solution can be given by the saddle point of the Lagrange function as:

$$\max_{\alpha, \nu} \min_{\Theta, b, \xi} L(\Theta, b, \xi; \alpha, \nu) \tag{2.30}$$

By applying partial derivative to each parameters:

$$\begin{cases} \dfrac{\partial L}{\partial \Theta} = 0 \to \Theta = \sum_{k=1}^{N} \alpha_k y_k \Theta(x_k) \\ \dfrac{\partial L}{\partial b} = 0 \to \sum_{k=1}^{N} \alpha_k y_k = 0 \\ \dfrac{\partial L}{\partial \xi} = 0 \to 0 \le \alpha_k \le C,\ \forall k \end{cases} \tag{2.31}$$

And substitution in formula 2.30, we can obtain:

$$\begin{aligned} &\max_{\alpha} W(\alpha) = \sum_{k=1}^{l} \alpha_k - \frac{1}{2} \sum_{k,l=1}^{l} \alpha_k \alpha_l K(x_k, x_l) \\ s.t. &\sum_{k=1}^{l} \alpha_k y_k = 0 \\ &0 \le \alpha_k \le C, \forall k \end{aligned} \tag{2.32}$$

where K(x_k, x_l) is a composite kernel called quasi-linear kernel, defined by

$$\begin{aligned} K(x_k, x_l) &= \Theta^T(x)\Theta(x) \\ &= (1 + x_k^T x_l) \sum_{j=1}^{M} R_j^T(x_k) R_j(x_l), \forall k, l \end{aligned} \tag{2.33}$$

From formula 2.32, could solve all the parameters in quail-linear kernel SVM which is all α_k. So, the final formulation of quasi-linear kernel SVM could write as:

$$y = sign\,[f(x)] \tag{2.34}$$

Where $f(x)$ could write as:

$$f(x) = \sum_{k=1}^{N} \alpha_k y_k K(x, x_k) + b \tag{2.35}$$

where positive constant α_k is Lagrange multiplier obtained from the QP optimization. The x_i for which $\alpha_k > 0$ are called support vectors which contribute to the geometric location of the separation boundary.

2.3.2 Quasi-Linear Kernel

As can be seen from the mathematical formula 2.33, quasi-linear kernel function can be seen as an explicit nonlinear mapping of the inner product representation. By adjusting the parameter M, that is, changing the number of local linear models, so the mapping performance of quasi-linear kernel function can be adjusted. Furthermore, $K(x_k, x_l) = \sum_{j=1}^{\infty} R_j^T(x_k) R_j^T(x_l)$ is a nonlinear kernel function with the following properties (see figure 2.5).

Corollary 1 *The positive definite of the quasi-linear kernel function: Since the quasi-linear kernel function is an inner product representation of an explicit nonlinear mapping, it satisfies the Mercer's theorem, so it is a positive definite kernel function.*

Corollary 2 *Modulation function of quasi-linear kernel function: The quasi-linear kernel function can adjust the complexity between the linear kernel function and the nonlinear kernel function by adjusting the parameter M. When $M = 1$, the quasi-linear kernel function degenerates into a linear function. When $M \to \infty$, the quasi-linear kernel function becomes a product of a linear kernel function and a nonlinear kernel function, so it is a nonlinear kernel function. Figure 2.5[11] shows this relationship.*

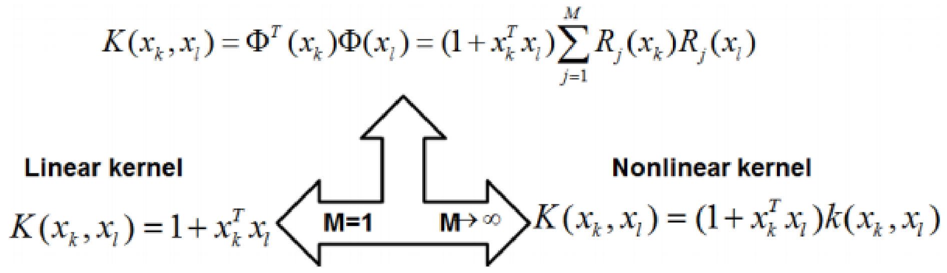

$K(x_k, x_l) = \Phi^T(x_k)\Phi(x_l) = (1 + x_k^T x_l)\sum_{j=1}^{M} R_j(x_k)R_j(x_l)$
Linear kernel
$K(x_k, x_l) = 1 + x_k^T x_l$
M=1
M→∞
Nonlinear kernel
$K(x_k, x_l) = (1 + x_k^T x_l)k(x_k, x_l)$

Figure 2.5 Quasi-linear kernel function adjust the number of local linear classifiers.

The quasi-linear kernel function can be constructed by introducing the local threshold basis function $R_j(x)$. When considering a radial basis function (RBF) as the basis function, we have the

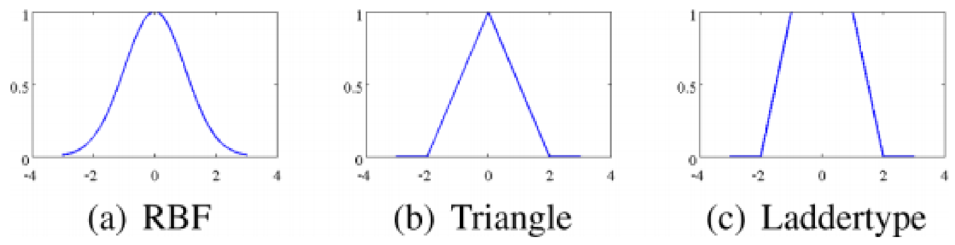


Figure 2.6 Different type of basis threshold functions.

following expression:

$$\widetilde{R}_j(x) = e^{-\frac{(x-\mu_j)^2}{\lambda \sigma_j^2}}, \; R_j(x) = \frac{\widetilde{R}_j(x)}{\sum_{j=1}^{M} \widetilde{R}_j(x)} \tag{2.36}$$

Where μ_j and σ_j can be regarded as the center and radius of the threshold basis function, λ can adjust the activation range of the threshold basis function in some ability. From the experimental results we can see, when the data were normalized, λ value can just be set to 1.

Similarly, as shown in figure 2.6 [11], triangular and trapezoidal functions can also be used as a threshold function, the drawback is the lack of smoothness. In the remainder of this thesis, the radial basis function (RBF) is default used as the local threshold basis function for the construction of quasi-linear kernel function.

2.3.3 Localization of Feature Space

From the derivation of the previous sections, it is shown that a quasi-linear kernel function is obtained by multiple local linear models with their corresponding local threshold basis functions. Therefore, before constructing a quasi-linear kernel function, we need to perform a local partition processing of the feature space to obtain the local information to construct each parameter in the threshold basis function.

The clustering technique can automatically decide the partition of local feature space according to the distribution relation of specific training data. B. Zhou uses a two-step approach to determine the final local partitions [11] (figure 2.7).

- **Step 1** According to the boundary detection algorithm, the data points located around the potential hyperplane are selected as the key optimization objects of the local partitioning algorithm, meanwhile, it avoids the extracted local threshold basis function deviates from the potential boundary hyperplane.

- **Step 2** Using the modified K-means clustering algorithm to localize the feature space, and avoid local sample unbalance.

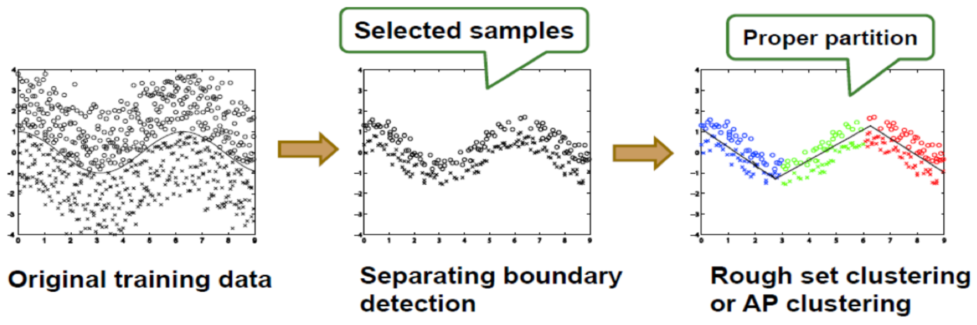

Selected samples
Proper partition
Original training data
Separating boundary detection
Rough set clustering or AP clustering

Figure 2.7 Modified k-means method on an artificial data set.

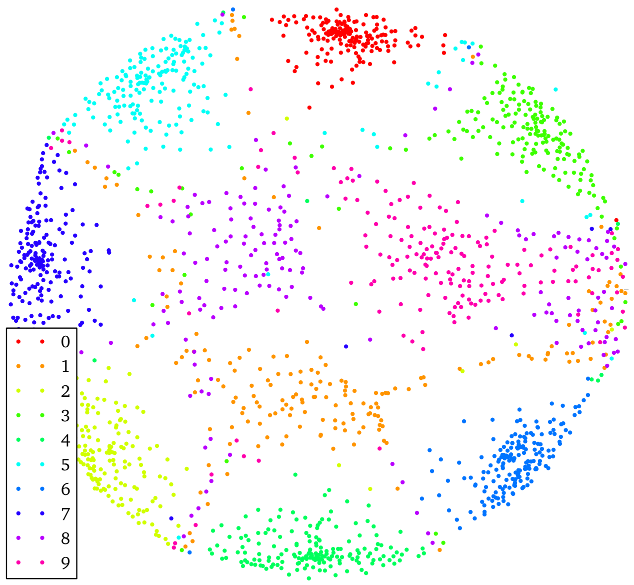


Figure 2.8 Proximity plot for a 10-class handwritten digit classification task.

But, In the process of construction of quasi-linear kernel function, the unsupervised learning method has a major flaw. Because of the characteristics of the clustering algorithm, it is difficult to deal with the problems like figure 2.8.

Chapter 3

Local Linearity Extraction Based on Samples Space Division

In the last chapter, the theory and formula derivation of quasi-linear support vector machines are introduced, and the characteristics of quasi-linear kernel functions are illustrated by examples. By analyzing the modified k-means method, some requirements of the constructed a good data-driven quasi-linear kernel functions are summarized.

1. Partitioned local linear feature space requires the balance of data distribution as much as possible.
2. The center of the threshold function corresponding to partitioned linear feature space should be as close as possible to the potential classification hyperplane.
3. The number of local linear feature spaces should be obtained automatically by the algorithm, rather than artificially pre-specified.

In this chapter, in order to facilitate the easy understanding, first of all, give an overview of the implementation framework with the help of some illustrative figures, and then discuss in detail of the use of decision trees to identify classification boundaries with local linear partitions. Finally, from the theory of statistical learning, discuss the usefulness and feasibility of extending the decision tree to decision forest, and give a concrete example.

3.1 Implementation Framework

In this part, we give an overview of the implementation framework with the help of some illustrative figures. Some of the implementation details would be introduced in the following sections. Here is the flow:

1. In order to first detect the nonlinear classification boundary (Remark 1), we use the fact, even a complexity boundary shape, the small enough local space is approximately linear boundary. According to this idea, we divide the input space of data into small enough space according to some rules, where the data in each subspace can be divided by a linear boundary. Finally, all of the linear boundaries could line up to get a non-linear boundary. In addition, since we already know the label information of all data, the classification result in current subspace can be used as one of the criteria for judging whether or not the division of the subspace is sufficient. Another criterion for determining how to divide the current space is to compute the distance of the data in the space from its center.

2. After solving how to get the local linearity (Remark 1), we will use ensemble learning method combine with hierarchical cluster algorithm to finally get a set of local linear linearity, in that set, each one includes the final number of local linear classifiers and their function scopes, which is later embedded into the composition of quasi-linear kernel (Remark 2). Finally, we could use valid data to determine which one is the best.

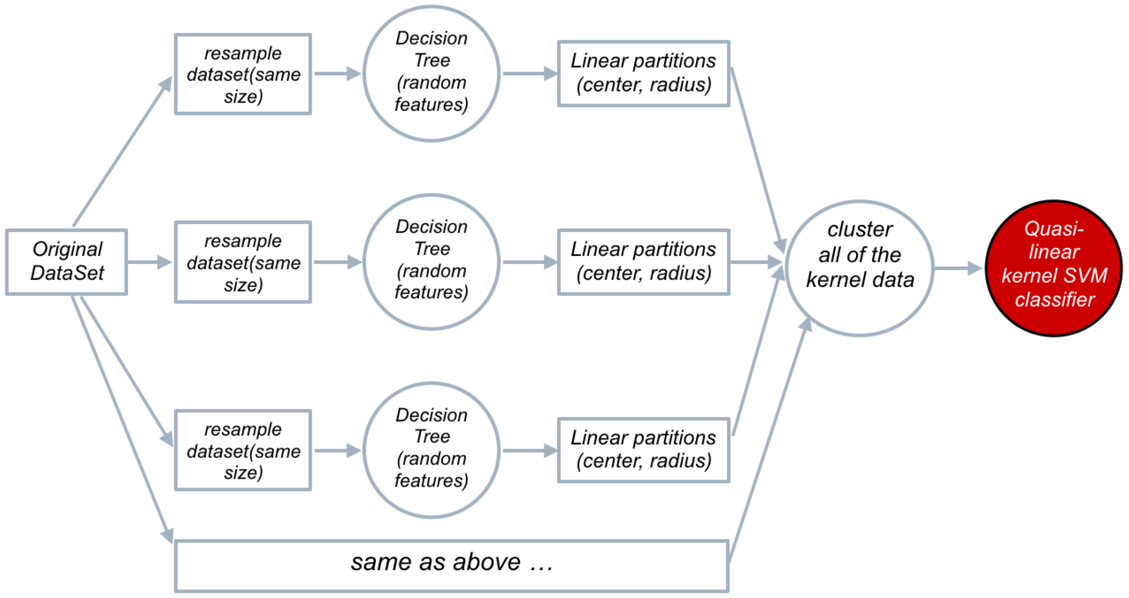

Original DataSet
resample dataset(same size)
Decision Tree (random features)
Linear partitions (center, radius)
same as above …
cluster all of the kernel data
Quasi-linear kernel SVM classifier

Figure 3.1 The framework of the proposed method .

From the figure 3.1 we could see that the basic component of this frame is decision tree, other technique just service as improving the precise of extracted local linearity by the decision tree.

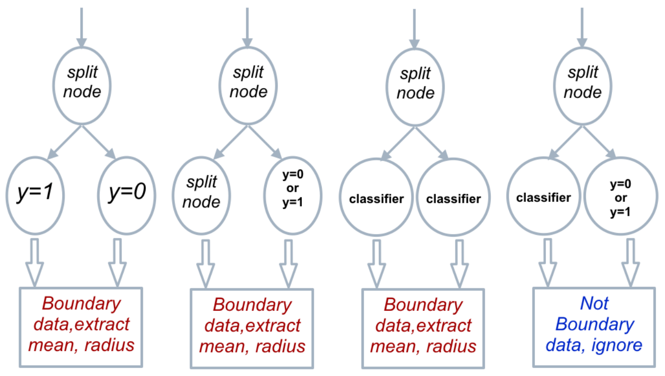


Figure 3.2 The extract condition of linear information in leaf nodes .

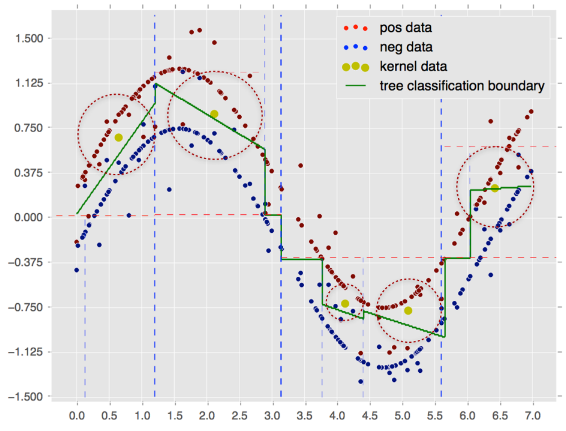


Figure 3.3 The extract linear information at leaf nodes which are linear classifiers.

In figure 3.2, we conclude four conditions when node is split. If the two children sample space is the potential boundary, then the algorithm will calculate the mean and radius corresponding to its sample space. Followed by this algorithm, figure 3.3 shows the final generated tree. In that, the green line is the classification boundary respect to the tree, the red and blue dotted line is corresponding each split feature value, deeper color means closer to the root node. Final, the yellow points are the mean value of the corresponding sample space, the surrounded circle is its radius.

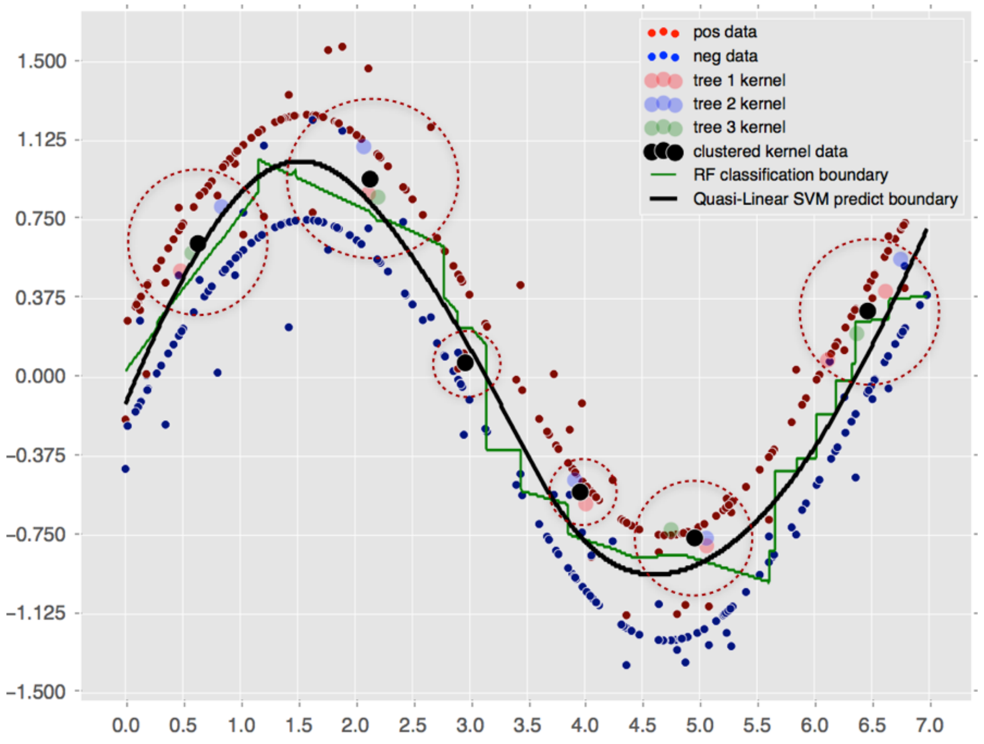


Figure 3.4 The final quasi-linear SVM trained by random forest.

Figure 3.4 give us a final classification boundary of trained quasi-linear kernel SVM which is the black solid line. before getting that line, we cluster all three decision trees' linear information, then get the black points which are the final local linear information. We also could see, compare with the figure 3.3, the classification boundary combined with three decision trees is more robust and smooth than only one tree.

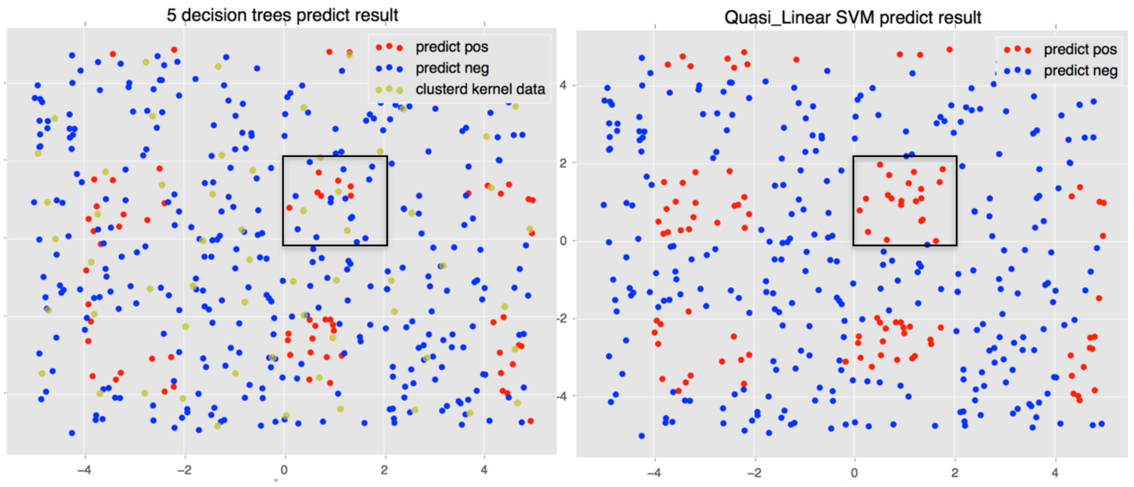


Figure 3.5 The final quasi-linear SVM trained by random forest compare with random forest predict result.

Moreover, in figure 3.5, each block is all same label, this example shows the final quasi-linear SVM trained by random forest has better performance than the random forest predict result. Since the final classification boundary has been globally optimized under the maintenance of the training data geometric shape.

To summary, the whole training frame for quasi-linear kernel SVM can be summarized:

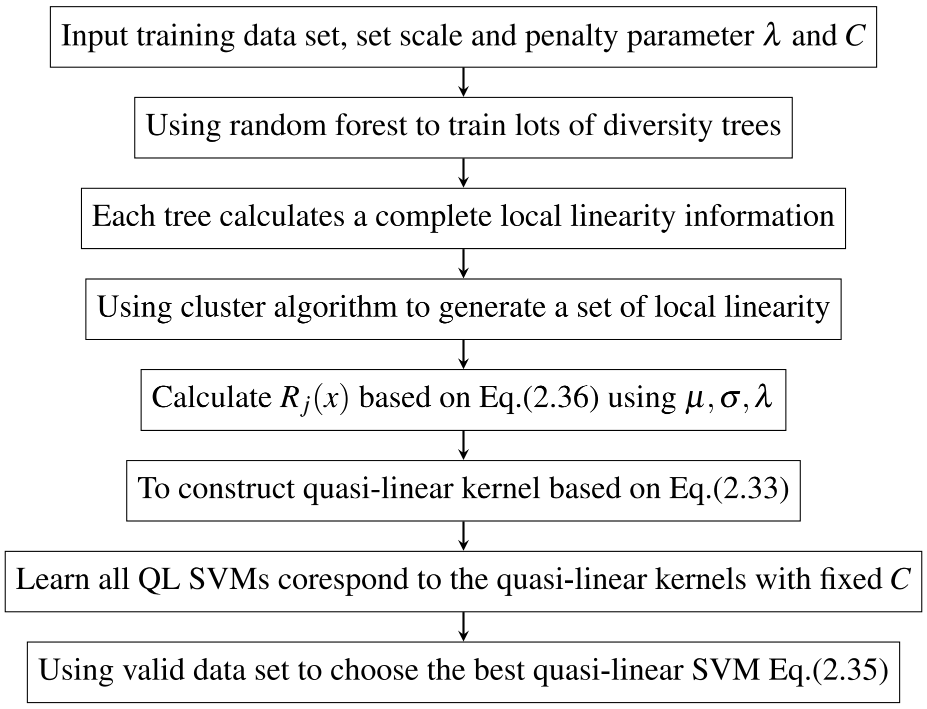

Input training data set, set scale and penalty parameter λ and C
Using random forest to train lots of diversity trees
Each tree calculates a complete local linearity information
Using cluster algorithm to generate a set of local linearity
Calculate $R_j(x)$ based on Eq.(2.36) using μ, σ, λ
To construct quasi-linear kernel based on Eq.(2.33)
Learn all QL SVMs corespond to the quasi-linear kernels with fixed C
Using valid data set to choose the best quasi-linear SVM Eq.(2.35)

Figure 3.6 Flow diagram of training process

In the test stage, the work flow as below:

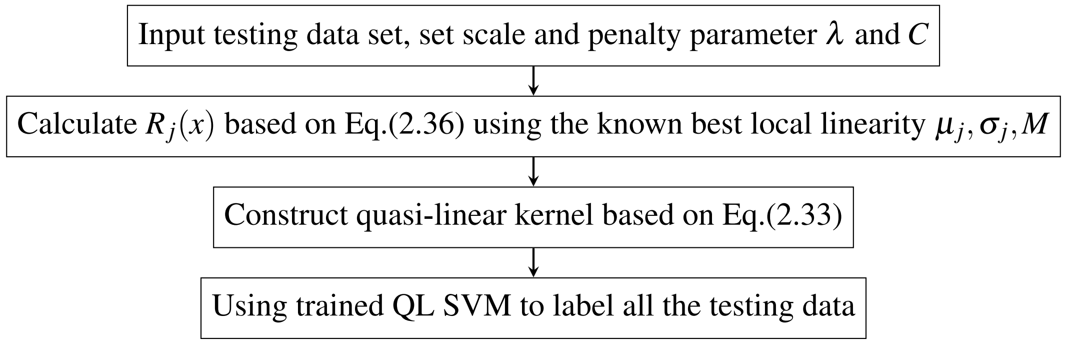

Input testing data set, set scale and penalty parameter λ and C
Calculate $R_j(x)$ based on Eq.(2.36) using the known best local linearity μ_j, σ_j, M
Construct quasi-linear kernel based on Eq.(2.33)
Using trained QL SVM to label all the testing data

Figure 3.7 Flow diagram of testing process

3.2 Tree Structured Models

In this section, we focus on understanding the principle of the decision tree model. The unifying work of Breiman [21], later complemented with the work of Quinlan [22], have set a series of tree model into a simple and consistent method framework, so, the tree model is gradually widely understood and used. Most importantly, decision trees are the basic component of many moderns and state-of-the-art algorithms, including random forest, which is described later.

As we will explore in detail later, the success of the decision tree model and the extension model are due to several factors that make it attractive for practical use:

- Decision trees are a non-parametric model. They can model any complex relationship between the input signal and the output signal without any a prior assumptions.
- Decision trees can handle multiple types of data, such as discrete or continuous variables, or a mixture of the two.
- Decision tree actually performs feature selection at the split, which makes the model more robust to irrelevant features or noise variables.
- Decision trees have the robustness to outliers or errors in labels.
- Decision trees are very easily to understand its processing.

From a geometric point of view, the principle of tree structured models is very simple. It consists of dividing the input space X into subspaces recursively, and then assigning constant predictive value $\widehat{y} \in$ Y to all objects $\mathbf{x}$ within each terminal subspace. For the sake of clarity, let us first define the following concepts:

Definition 1 *A tree is a* directed graph $G = (V, E)$*, in which all edges are from the root node, and any path from root node to the termination node is connected unique.*

Definition 2 *If there exists an edge from t_1 to t_2 (i.e., if $(t_1, t_2) \in E$) then node t_1 is said to be the* parent *of node t_2 while node t_2 is said to be a* child *of node t_1.*

Definition 3 *In the tree diagram, if a node has one or more child nodes, it is called an internal node; If it has no children node, it is called terminal node or leaf node.*

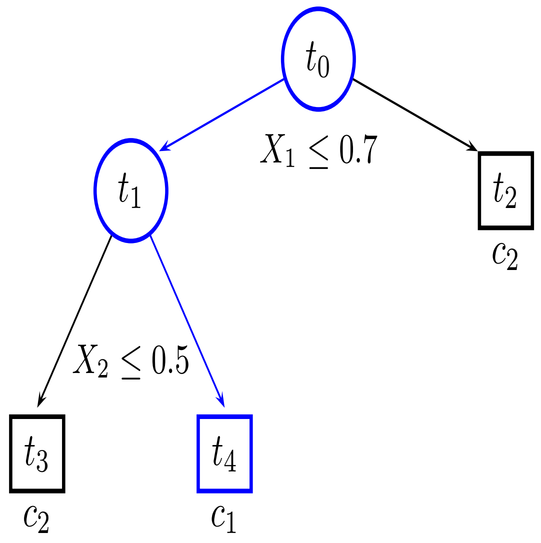
Definition 4 *A* binary tree *is a rooted tree in which all internal nodes have up to two children.*

Similarly, the decision tree is also a set of *if-then* rules. The decision eigenvalues of each node in the path from the root node to the leaf node corresponding to the split condition and the class of the leaf node is the conclusion of the rules.

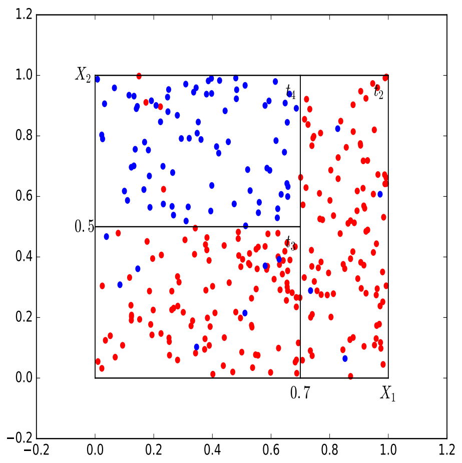
So, in one word, a *tree-structured model* of *decision tree* can be defined as a non-linear transformation $\varphi : X \mapsto Y$, where any node t represents a sub feature space $X_t \subset X$ of the input feature space, with the root node corresponding to input feature space X itself. Internal nodes n are labeled with a *split* s_t taken from one of the feature space. It divides the feature space X_t that node t represents into disjoint subspaces respectively corresponding to each of its children nodes.

Prediction of the output value $\widehat{y} = \varphi(\mathbf{x})$ in a decision tree.

```
1: function PREDICT($\varphi, \mathbf{x}$)
2:     $t = t_0$
3:     while $t$ is not a terminal node do
4:         $t =$ the child node $t'$ of $t$ such that $\mathbf{x} \in X_t$
5:     end while
6:     return $\widehat{y}_t$
7: end function
```



(a) A decision tree φ built for a binary classification problem from an input space $X = [0,1] \times [0,1]$.



(b) Partition of X induced by the decision tree φ into subspaces.

As an example, figure 3.8(a) illustrate a decision tree φ made of five nodes and partitioning the input space $X = X_1 \times X_2 = [0;1] \times [0;1]$ for a binary classification problem (where $Y = \{c_1, c_2\}$). Figure 3.8(b) visualize its data space, blue dots correspond to objects of class c_1 while red dots correspond to objects of class c_2.

3.2.1 Splitting Rules

The decision tree is essentially a set of classification rules derived from the training data set. There may or may not be any decision trees matching the training data set. From another point of view, decision tree model is conditional probability model estimated from the training data set. There are an infinite number of conditional probability models for classes based on feature space partitioning. The conditional probability model we choose should not only fit well the training data but also have a good prediction for the unknown data.

The decision tree learning process also uses the loss function to represent this goal like others. As described below, the loss function of decision tree learning is usually a regularized maximum likelihood function. The decision tree learning strategy is to minimize the loss function as the objective function. When the loss function is determined, the learning problem becomes the problem of choosing the optimal split feature value under the loss function. Since the choosing optimal decision tree from all the possible decision trees is the NP-complete problem, the decision tree learning algorithm actually usually adopts the heuristic method to approximate the optimization problem. Therefore, the resulting decision tree is *sub-optimal*. Formally, the decrease of impurity of a binary split s is defined as follows:

Definition 5 *The* loss decrease *of a binary split $s \in S$ dividing node t into a left node t_l and a right node t_r is*

$$\Delta i(s,t) = i(t) - p_l i(t_l) - p_r i(t_r) \tag{3.1}$$

where p_l(resp., p_r) is the proportion $\frac{N_{t_l}}{N_t}$ (resp., $\frac{N_{t_r}}{N_t}$) of learning samples from L_t going to t_l (resp., to t_l) and where N_t is the size of the subset L_t.

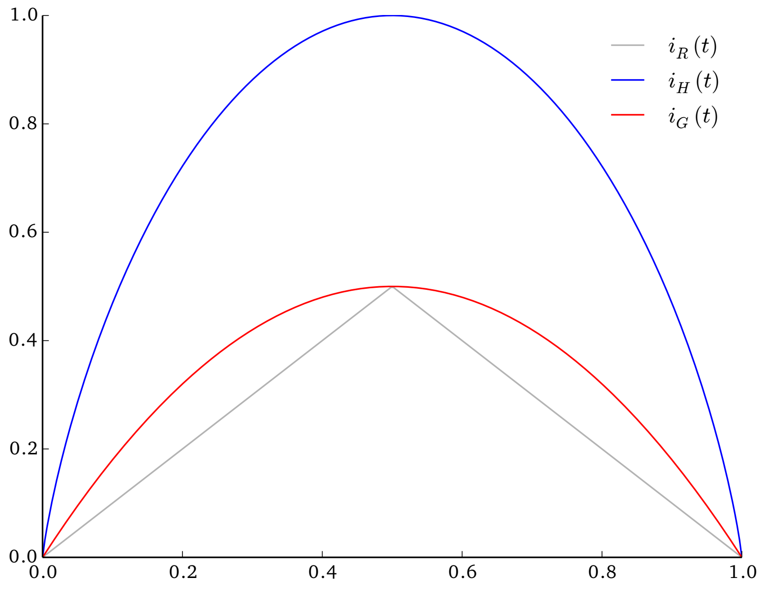


Figure 3.8 The relationship in entropy, Gini index and classification error rate

For purpose of comparison, we illustrate three criteria in loss functions for feature selection, as in figure 3.8, where $i_R(t)$ is classification error rate, $i_H(t)$ is entropy, $i_G(t)$ is gini index.

Decision tree learning algorithm is a process of usually a recursive selection of optimal features, and according to the selected feature value do train data splitting, so that each sub-data set has the best classification process. This process corresponds to the division of the feature space, also corresponds to the decision tree construction process. This is done until all the training data subsets are classified correctly, or there are no features that can be used to divide.

The process of splitting decision tree is actually the choice of classification features. Important features have a large impact on the outcome of the classification, while irrelevant features have no significant impact on the results. For the decision tree, the feature selection is to decide which feature to use to partition the feature space.

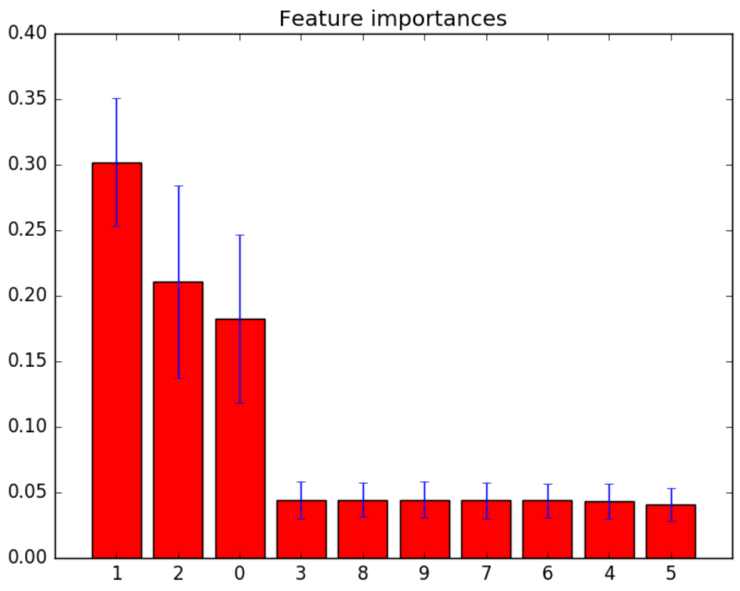


Figure 3.9 evaluate the importance of features by random forest.

Figure 3.9 shows the use of forests of trees to evaluate the importance of features on an artificial classification task. The red bars are the feature importances calculated by random forest (later discuss) along with their inter-trees variability. As expected, the plot suggests that 3 features are informative, while the remaining are not.

3.2.2 Generation of Classification Trees

In this section, we introduce common decision tree generation algorithms, namely classification and regression tree (CART) model [21]. CART is a learning method that outputs the conditional probability distribution of random variable Y under given input random variable X. CART assumes that the decision tree is a binary tree and the values of the internal node feature are "yes" and "no". Such a decision tree is equivalent to recursively bisecting each feature, dividing the input feature space into finite cells, and confirm the prediction probability distribution on these cells. In our method, the decision tree here based on CART model to modify.

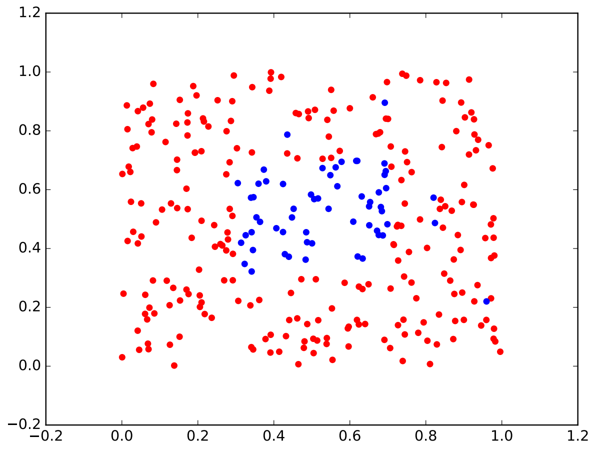


Figure 3.10 Binary classification task.

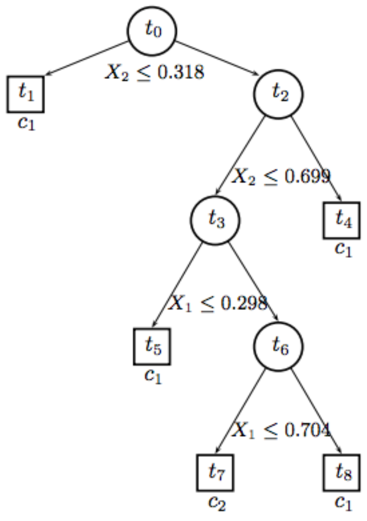


Figure 3.11 Decision tree learned from Figure 3.10.

t	0	1	2	3	4	5	6	7	8
left_child	1	–	3	5	–	–	7	–	–
right_child	2	–	4	6	–	–	8	–	–
feature	2	–	2	1	–	–	1	–	–
threshold	0.318	–	0.699	0.298	–	–	0.704	–	–
impurity	0.315	0.020	0.409	0.497	0.047	0.	0.4666	0.	0.107
n_samples	300	98	202	120	82	31	89	54	35
value	241/59	97/1	144/58	64/56	80/2	31/0	33/56	0/54	33/2

Table 3.1 Array representation of the decision tree in Figure 3.11.

As an example, Table 3.1 shows the array representation of the classification tree which uses Gini index as criteria in loss function.

It can be known that the deeper the tree, the smaller the split feature space of the input feature space is. Obviously, with the exponential growth of leaf nodes, will lead to a dramatic increase in the complexity of the model, finally, the final model will fit the noise data, resulting in over-fitting occurs. Therefore, the decision tree algorithm usually has pruning operations, contains pre-pruning and post-pruning. We will solve this problem by introducing random forest technique in the latter section.

3.3 Local Linearity Extraction Based on Decision Tree

In this section, we will discuss the core idea of this proposed method, namely extract local linearity by decision tree model. In the relative work, Schulter gives us a method how naturally doing a locally linear multivariate regression problem and that random forests nicely fit into this framework [23]. In their work, they propose a novel regularized objective function optimized during tree growing that operates not only considering the best split feature space but also the best split data space.

Training a single tree involves recursively splitting the training data into disjoint subsets by finding splitting functions for all internal nodes in tree.

$$\sigma(x_j, s^*) = \begin{cases} L_{t_l} \text{ if } r_{s^*}(x_j) < v'_k \\ L_{t_r} \text{ otherwise} \end{cases} \tag{3.2}$$

Inspired by them, we propose a loss function considering the division of the input sample space with the classification error rate evaluation criteria, we get the final cost equation as below:

$$E(x_i) = \frac{1}{N} \sum_{n=1}^{N} \left(||y_i - \hat{y}_i||_2^2 + \lambda_1 \cdot ||x_i - \bar{X}||_2^2 + \lambda_2 \cdot ||x_i - \bar{X}_{y \neg i}||_2^2 \right) \tag{3.3}$$

Where, the first term is Mean Square Error (MSE), the second term is the distance of current data from the center of the data space, the third term is the distance of current data away from the opposite label data center.

The intuition of the 2,3-th terms is that we not only require to put samples in a leaf node that have right predicted label $\hat{y}_i$, we also want those samples themselves (i.e. x_i) to be cluster (compactness) as a simple shape so that these data could fall into the same leaf node, which potentially eases the task for the linear classification model $m_l(x_i)$ in the leaf. By the way, the 2,3-th terms in this formula are from Pearson Correlation, which is used to calculate the similarity of two vectors.

$$Corr(x,y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2} \sqrt{\sum (y_i - \bar{y})^2}} \quad = \frac{< x - \bar{x}, y - \bar{y} >}{||x - \bar{x}|| ||y - \bar{y}||} = CosSim(x - \bar{x}, y - \bar{y}) \tag{3.4}$$

After fixing the best split $\sigma(\cdot)$ according to Eq 3.3, the data in the current node is split and forwarded to the left and right children respectively. Growing continues until one of the stopping criteria is met and the final leaf nodes are created. In our method, The needed μ and λ in Eq.(2.36) are calculated in the datasets in the corresponding leaf node which is the boundary data, specific, we calculate the mean value and standard deviation as the μ and λ. For preciser result, this method could be improved.

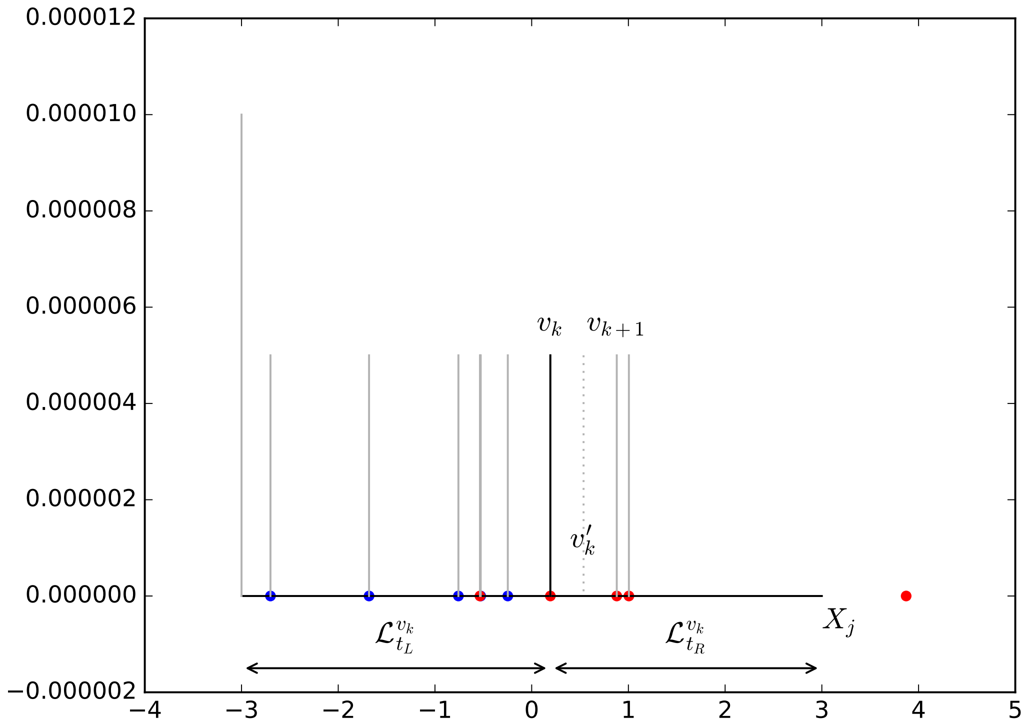


Figure 3.12 Binary partitions of L_t on the ordered variable X_j. Setting the decision threshold v to any value in $[v_k, v_{k+1}]$.

The important thing here is that assuming distinct input values for all $\mathrm{N_t}$ node samples, the number of partitions of L_t into k non-empty subsets is given by the Stirling number of the second kind [24]:

$$S(N_t, k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j} j^{N_t}, \tag{3.5}$$

which reduces to $2^{N_t-1} - 1$ for binary partitions. Given the exponential growth in the number of partitions, the naive strategy of enumerating all partitions and picking the best of them is often computationally intractable. For this reason, in our method, we will choose almost 66% percent of continuous feature values as candidate split point. We usually assumes that s^* is a good enough approximation, lives in a family $s^* \subseteq S$ of candidate splits of restricted structure (figure 3.12).

To summary, we lead to the following procedure:

Find the best split s^* that partitions L_t.

```
 1: function FindBestSplit($L_t$)
 2:     $\Delta = -\infty$
 3:     for $j = 1, \ldots, p$ do
 4:         Find the best binary split $s_j^*$ defined on $X_j$
 5:         if $\Delta i(s_j^*, t) > \Delta$ then
 6:             $\Delta = \Delta i(s_j^*, t)$
 7:             $s^* = s_j^*$
 8:         end if
 9:     end for
10:     return $s^*$
11: end function
```

3.4 A Random Forest Framework

From the previous section, we have explained that based on CART algorithm, by using our new loss function, we could use it to extract local linearity (figure 3.3). However, in order to ensure the generalization performance of the model, usually, the pruning operation must be carried out during the training process. But, we have a better way to fix this problem by naturally introduce decision forest. We say the decision forest is based on the ensemble learning theory. Briefly, the benefits are obvious:

- First, the generalization performance of the whole model is better.
- Secondly, no longer needs worry about the over-fitting problem, means we do not need prune operations.

Thus, the linear partition based on the combination of all of the decision trees will ensure that the constructed quasi-linear kernel function has more precise data shape information. So, the final trained QL-SVM could improve its performance.

This section is organized as follows: first, introduce the basic theory of ensemble learning method, explain its characteristics and the place need to pay attention to which it is implemented; Then introduce Random Forest learning framework; finally, we discuss how to use the random forest [25] to extract the segmentation linearity of the input sample with clustering algorithm.

3.4.1 Ensemble Learning Methods

Ensemble learning is accomplished through the construction and integration of multiple learners to complete the learning task, sometimes called multi-classifier system. The general structure of ensemble learning:

1. Generate a group of individual learners and then combine them with some strategy, in which the individual learner is usually an existing learning algorithm, such as CART.

2. The integrated individual learners are the same type, such as all of the decision tree, this integration is called homogeneous. Relatively, if the individual learners are different, called as heterogeneous.

Ensemble learning usually could obtain better generalization performance than the single learner, especially when the individual learner is a weak learner. In addition, when strong learners are used, generally less individual learner integration can achieve better generalization performance.

$$H(x) = \text{sign}\left(\sum_{i=1}^{T} h_i(x)\right) \tag{3.6}$$

Eq. 3.6 says that the ensemble method combines a number of T of basis classifiers with a simple voting method, if more than half of the basis classifiers are correct, then the integration classification is correct.

Assuming that the error rates of the basis classifiers are independent of each other, the Hoeffding inequality shows that the integrated learning error rate is:

$$\begin{aligned} P(H(x) \neq f(x)) &= \sum_{k=0}^{\lfloor T/2 \rfloor} \begin{pmatrix} T \\ k \end{pmatrix} (1-\varepsilon)^k \varepsilon^{T-k} \\ &\leq \exp\left(-\frac{1}{2}T(1-2\varepsilon)^2\right) \end{aligned} \tag{3.7}$$

From formula 3.7 we could see that with the increase in the number of base learners, the ensemble error rate will exponentially decline, and ultimately tend to zero.

However, the individual learners must match with some requirements. if expected to get good training result, it is necessary that all of the base learners as different as possible while they have somehow good performance. In fact, if all the base learner input data are the same, then the above request is very contradictory, so the core research of ensemble learning is focused on to solve this problem ([26, 27, 28]). Finally, there are two types of combination strategies: average method, "learning method".

Remark 3 *Ambiguity decomposition For prediction averaging, as defined in Equation 3.6, the ambiguity decomposition guarantees the generalization error of the ensemble to be lower than the average generalization error of its constituents [26]. Formally, the ambiguity decomposition states that*

$$E = \overline{E} - \overline{A} \tag{3.8}$$

where $\overline{E} = \sum_{i=1}^{T} w_i E_i$ and $\overline{A} = \sum_{i=1}^{T} w_i A_i$, the first term is the average generalization error of the individual models, the second term is the ensemble ambiguity and corresponds to the variance of the individual predictions around the prediction of the ensemble. Since $\overline{A}$ is non-negative, the generalization error of the ensemble is therefore smaller than the average generalization error of its constituents. Formula 3.8 clearly points out that the higher the accuracy of individual learners, the greater the diversity, the better integration performance could be achieved.

3.4.2 Random Forest

One of the ensemble models, called Random Forest, is a parallel model, in which there is no dependency between individual learners, and learning tasks can be performed at the same time [25]. As we can see from the previous subsection, for the strong generalization performance, the individual learners in the ensemble should be as independent as possible. Although pure independence can not be achieved, we can try to make the base learner as different as possible. Random forest (RF), a word summary, is the combination of many diversity decision trees. It improves the diversity of the trees in the following way:

- The input data is adjusted first, using the Bagging method [27], which is based on bootstrap sampling. Specifically, the flow is as follows: for a given data set containing m samples, we first randomly draw one from it into the sample set, and then put the sample back to the initial data set, so that in the next sampling process the same one still has a chance to be chosen. Now, after m times re-sampling process, we can get a sample set with m samples, in which some samples in the initial training set appear many times in the sampling set, while others never appear. From formula 3.6, about 63.2% of the samples in the initial training set appear in the sampling set. In this way, we can get a number of T sample sets with each has m training samples. Now, we could train a decision tree based on each sample set and then combine these decision trees.

$$\lim_{m \mapsto \infty} \left(1 - \frac{1}{m}\right)^m \mapsto \frac{1}{e} \approx 0.368 \tag{3.9}$$

- Random forest introduces stochastic feature selection in the training process of the decision tree. In the traditional decision tree, the optimal feature is selected in the feature set of the current node. In RF, at the current node, firstly, a subset of k features is selected at random, and then an optimal feature is selected from the subset for splitting process. Here, the parameter k controls the degree of randomness: if $k = d$, the construction of the decision tree is the same as that of the traditional decision tree; if $k = 1$, then a feature is chosen randomly for splitting; in general, The recommended value $k = log_2 d$.

For simple analysis, If the computational complexity of a single decision tree is $O(m)$, $O(s)$ is the complexity of the sampling and integration process, So train a random forest is almost $T(O(m)+O(s))$, with the same order of complexity compares with direct use of a decision tree. Therefore, the random forest is simple, easy to implement, computational overhead is small, but it shows a strong performance in a lot of real tasks.

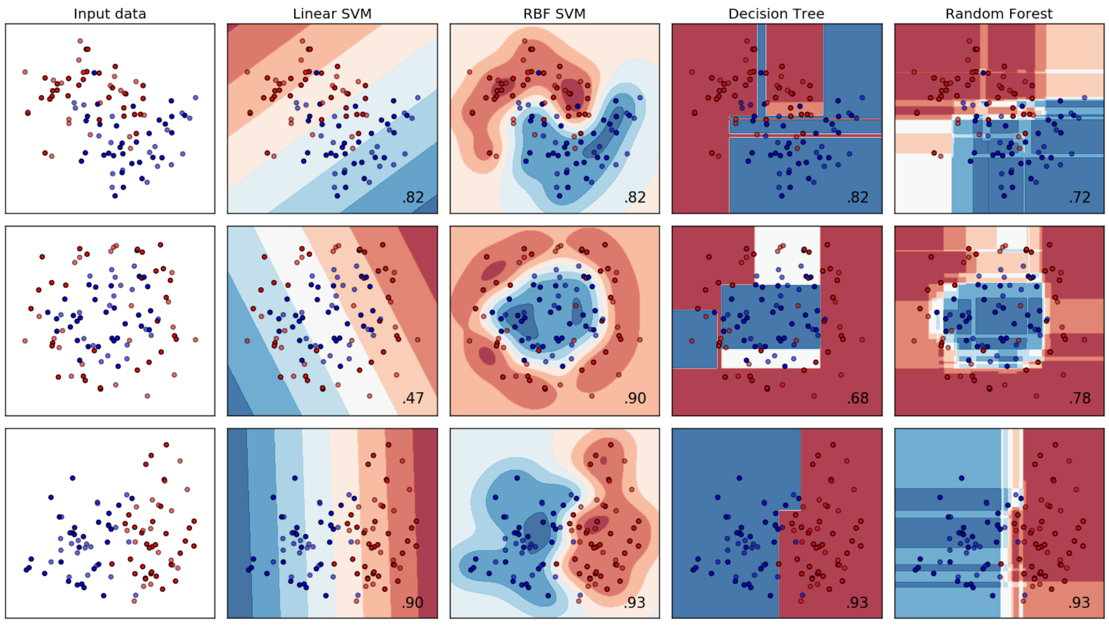


Figure 3.13 A comparison of classification boundaries of a several classifiers on synthetic datasets.

Figure 3.13 is a comparison of a several classifiers on synthetic datasets. The plots show training points in solid colors and testing points semi-transparent. The lower right shows the classification accuracy on the test set. From figure 3.13 we can conclude that RF could use combination of all classification boundaries corresponding each decision tree get a good non-linear separation boundary.

To sum up how to enhance diversity. The general idea is to introduce randomness in the learning process. The common ways are to disturb the data samples, input features, output results or model parameters. In RF, we use the data disturb and features disturb.

- The data disturb makes only about two-thirds of the original training samples appear in the training samples of each tree, and some of the samples are repetitive. From the perspective of the boosting learning method, we can say that this is equivalent to re-weighting the input samples. The weight of the data does not appear as 0, appeared once as 1, appeared twice as 2, and so on.

- RF also uses the features disturb, we know that the split process is actually a process of feature selection, and in RF each time only considers the subset of features. So with a deeper understanding, the splitting process of the decision tree first maps the feature space to some sub-feature space, next selects one.

- Furthermore, if the parameters of each tree in the random forest are mixed with randomness, this will make the prediction hypothesis space of each tree become more different, namely, the model complexity of the each tree is quite different.

3.4.3 Cluster Local Linear Partitions

In this paper, each decision tree will get a complete set of local linear information, but the cause of the performance of the model itself, this set of information is not accurate. However, it is sure that this set can express the approximate geometric information of the training data. In order to obtain more accurate local linear information, firstly we obtain several different sets of local linear information by random forest algorithm which uses lots of diversity decision trees; next, to obtain a set of reliable local linear information, hierarchical clustering algorithm is used, which can ensure that the extracted geometric information is not lost during the clustering process.

In this section, we first give the hierarchical clustering algorithm's diagram, explain its working principle, and then give the local linear partitions graph after clustering.

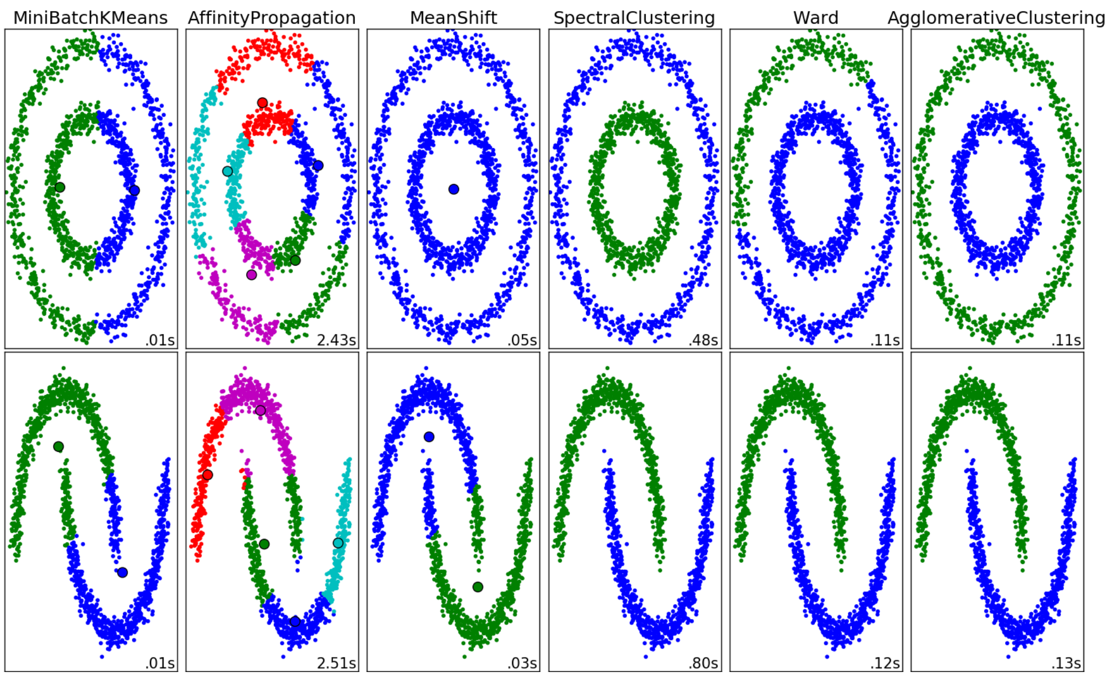


Figure 3.14 Show characteristics of different clustering algorithms.

Figure 3.14 aims at showing characteristics of different clustering algorithms. We can see that spectral clustering and agglomerative clustering could keep the data structure when doing clustering operation. So, In our method, we use agglomerative cluster do the local linear information cluster.

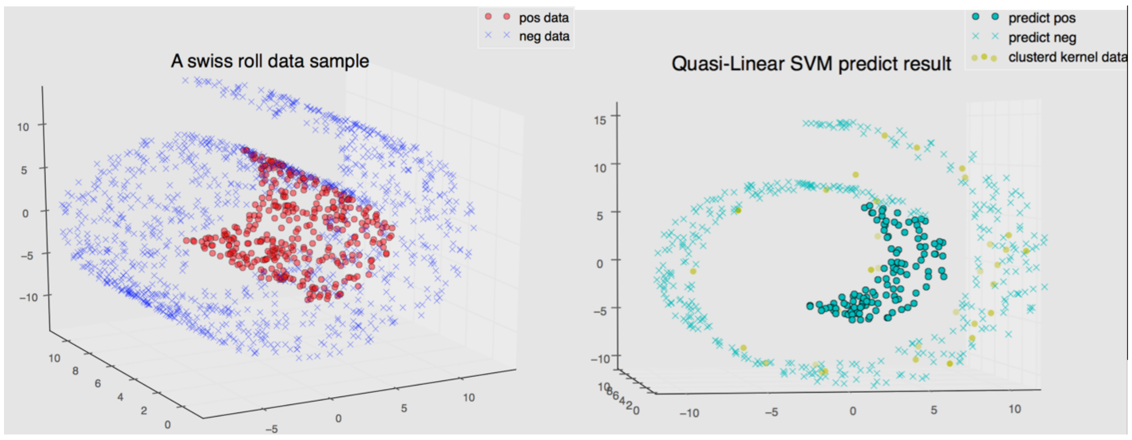


Figure 3.15 Proposed method handle with non-linear data set

From figure 3.15, we can know that by keeping geometric information during local linearity cluster process, final trained quasi-linear SVM still a good non-linear classifier, so that could handle with non-linear data set like figure 2.8.

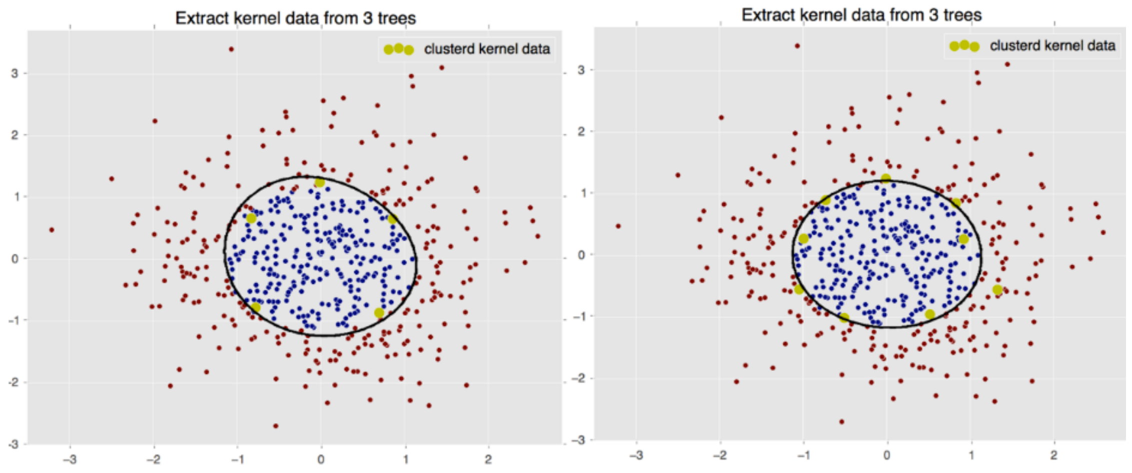


Figure 3.16 Proposed method choose variable number of local linear classifiers after training random forest.

From figure 3.16, we can say that our method is flexible enough when choose the number of local linear classifiers. This property is very import for constructed quasi-linear kernel performance (from Remark 2).

Chapter 4

Experiments

In this section, we evaluate our algorithm on real-world data sets, the results as shown in the following figures and tables. In our experiments, the LIBSVM tool developed by Chang and Lin [29] is taken as a basis and all experiments are conducted on a Windows 7 machine with a $1.8GHz$ CPU and $8.0GB$ of RAM.

4.1 Evaluation Metrics

Since we focus on the improvement of classifier performance on binary classification, for data sets containing more than two classes, we use one-versus-all strategy to transform a multi-class problem to multiple binary classification problems. In addition, we use L2 normalization method to pre-process data.

In section 4.2, we use *accuracy* as the metric to examine the performance comparison between the quasi-linear kernel and other kernels and Random Forest. In section 4.3, we have a data sets characterized by high-noise, in order to provide a comparable result, these three indicators are specifically designed for binary classification problems to be used, which is called *precision*, *recall* and $F-1$ score. Conceptually, the *precision* represents the proportion of the positive samples to the samples of all the positive samples. The *recall* represents the proportion of the positive samples to the original samples. The four evaluation indicators can be calculated by the following form:

$$\begin{aligned} accuracy &= \frac{TN+TP}{TN+TP+FN+FP} \\ precision &= \frac{TP}{TP+FP} \\ recall &= \frac{TP}{TP+FN} \\ F-1 &= \frac{2\times precision\times recall}{Precision+recall} \end{aligned} \tag{4.1}$$

Where TP represents the number of positive class samples correctly judged as a positive class; FP represents the number of negative class samples judged as a positive class by mistake; FN represents the number of positive class samples that are wrongly judged as negative class.

4.2 UCI Data Set

In order to compare the classification accuracy of each algorithm on the actual data set, this section compares and summarizes the performance of each algorithm using 7 widely used data set from the UCI machine learning database [30]. In Table 4.1, we summarize the characteristics of the selected data sets such as the features, the sizes, the number of positive and negative samples.

Table 4.1 Description of the data sets

data sets	features	Positive	Negative	Total
Breast [31]	10	458	241	699
Glass	9	70	144	214
banknote	4	610	762	1372
Yeast-MIT [32]	8	244	1242	1486
Yeast-NUC	8	430	1056	1486
Yeast-CYT	8	464	1022	1486
Yeast-ME3	8	163	1323	1486
Sonar [33]	60	97	111	208
Arrhythmia [34]	279	245	207	452
Haberman	3	225	81	306

In the tree parameter selection, we set tree number is 10, the random feature size is $log_2 d$, the split threshold is linear classification error less than 5%, min samples in the leaf is 10. For the SVM relative parameter selection, we set $C \in \{0.1, 0.5, 1, 5, 10, 20, 50\}$ and $\gamma \in \{1\}$ for the quasi-linear SVM, and using a 3-fold cross-validation to train each quasi-linear SVM in the set kernels of each one has a different number of local linear classifiers, then choose the best one. On the other hand, a 3-fold cross-validation is used to estimate parameters like $\gamma \in \{0.1, 0.5, 1, 5\}$ for SVM with RBF kernel and $C \in \{0.1, 0.5, 1, 5, 10, 20, 50\}$ for SVM with RBF and linear kernels. For each data set, we randomly split the data set into two halves, one-half for training, the other for testing. Table 4.2 summarizes the mean accuracies and the standard deviations. The result is the average performance of 50 times of random trials.

In Table 4.2, it can be seen that the quasi-linear kernel function overcomes the performance of linear kernel function and the RBF kernel function on the most data sets, even though cross-validation of the hyperparameters is not performed (with the choice of a broader hyperparameter range, it is possible to make RBF kernel function to achieve better performance). This means that although the quasi-linear kernel function is designed to substitute RBF kernel function as a nonlinear model when the RBF kernel function is over-fitted, an effective potential boundary can be obtained by accurately designing the local threshold basis function, this prior knowledge can make quasi-linear kernel function more powerful than the traditional widely used kernel functions. Moreover, RF trained classification boundary is also a robust one, although it is not optimized in a global view.

Table 4.2 The classification accuracies of SVM with different kernels and RF

data sets	Linear Kernel	RBF Kernel	Quasi-Linear Kernel	Random Forest
Breast	87.6103±0.0132	89.0591±0.0139	89.1434±0.0027	**92.8236±0.0465**
Glass	74.4723±0.0424	78.8403±0.0366	**81.3139±0.0680**	79.4412±0.0472
banknote	98.7812±0.0032	99.8201±0.0017	**99.8532±0.0025**	99.2782±0.0045
Yeast-MIT	87.1837±0.0184	**88.3316±0.0088**	87.6221±0.0035	87.2820±0.0039
Yeast-NUC	74.8692±0.0134	75.3401±0.0136	**76.2881±0.0072**	73.4125±0.0341
Yeast-CYT	68.4178±0.0117	71.8059±0.0161	**74.3478±0.0097**	73.0211±0.0151
Yeast-ME3	91.8627±0.0072	92.7296±0.0071	**93.6181±0.0072**	90.1097±0.0149
Sonar	74.0067±0.0395	**82.6040±0.0459**	81.7396±0.0397	75.9682±0.0353
Arrhythmia	74.9270±0.0211	74.1361±0.0216	**76.5527±0.0156**	68.5072±0.0363
Haberman	73.1884±0.0247	73.0710±0.0196	**75.8682±0.0236**	71.2472±0.0266

4.3 Yeast Gene Data Sets

This yeast gene data set is always a multi-label task [8, 35], contains 625 gene sequence training samples and 328 test samples. Each sample has a feature representation of 478 dimensions and is a high-dimensional high-noise dataset. Several labels were selected and illustrated in Table 4.3, where we only list the number of positive and negative samples. The experimental parameters are consistent with the previous section. In Table 4.4, the *precision*, *recall* and $F-1$ are used as the evaluation metrics, and we highlight the $F-1$ score in each row correspond to the best one.

Table 4.3 Yeast Data Set Description

dataID	Pos.	Neg.	Data Description
14.01	61	564	protein folding and stabilization
14.03	158	467	protein targeting, sorting and translocation
14.07	306	319	protein modification
14.07.02	41	584	modification with sugar residues
14.07.02.01	11	614	O-directed glycosylation, deglycosylation
14.07.03	91	534	modification by phosphorylation, dephosphorylation
14.07.04	32	593	modification by acetylation, deacetylation
14.10	118	507	assembly of protein complexes
14.13	150	475	protein/peptide degradation
14.13.01.01	75	550	proteasomal degradation (ubiquitin/proteasomal pathway)

Table 4.4 Classification results on the Yeast Gene Data Sets

	SVM with linear kernel			SVM with RBF kernel			quasi-linear SVM		
dataID	P	R	F	P	R	F	P	R	F
14.01	0.3823	0.4814	0.4262	0.6923	0.3333	0.4500	0.4516	0.5185	**0.4827**
14.03	0.4144	0.5476	0.4717	0.3951	0.5833	0.4711	0.7500	0.3962	**0.5185**
14.07	0.7362	0.7052	0.7200	0.7388	0.7000	0.7189	0.7320	0.8052	**0.7669**
14.07.02	0.4761	0.4166	0.4444	0.5000	0.3750	0.4285	0.6250	0.4167	**0.5000**
14.07.02.01	0.7500	0.7500	**0.7500**	0.5000	0.2500	0.3333	0.7500	0.7500	**0.7500**
14.07.03	0.6571	0.4339	0.5227	0.6969	0.4339	0.5348	0.7320	0.5828	**0.6489**
14.07.04	0.1667	0.1000	0.1250	0.2000	0.1500	0.1714	0.2400	0.1714	**0.2000**
14.10	0.3541	0.2881	0.3177	0.3389	0.3389	**0.3389**	0.3333	0.3333	0.3333
14.13	0.2568	0.3943	0.3111	0.2916	0.4225	0.3448	0.3882	0.4647	**0.4230**
14.13.01.01	0.3888	0.2000	0.2641	0.3200	0.2285	0.2666	0.5625	0.2571	**0.3529**

It can be seen from the table 4.4 that $F-1$ scores obtained by using the linear kernel in most cases are higher than those using RBF kernel function. This phenomenon shows that RBF kernel function has a serious over-fitting problem. Therefore, the quasi-linear kernel can easily overcome these two kernel functions. The reason is that the quasi-linear kernel function is an extension of the linear function, so as long as we can get a good division of the data sample space (get the prior knowledge of the data shape) can naturally overcome the performance of linear kernel function. Secondly, because the RBF kernel is equivalent to mapping the samples from the original input space to an infinite dimension space, it has too strong fit ability to achieve good generalization performance in the high noise dataset. Therefore, the good performance of quasi-linear support vector machines on such data sets can be affirmed.

Chapter 5

Conclusions

To conclude, the major contribution of this paper is to provide a flexible and robust method to composite a quasi-linear kernel. The prior knowledge embedded in the quasi-linear kernel function is actually the basic geometric skeleton shape about the final classification boundary. By this reason, we call this kernel is a special data-dependent kernel. In the first step, a decision tree based approach is developed to detect local linear partitions, then the combination of ensemble learning and cluster method, optimizing the detected local linearity so as to construct an informative quasi-linear kernel. In the second step, by using the trained kernel, a nonlinear classifier consisting of multiple local linear classifiers is built in an exactly same way as a standard SVM so as to get a globally optimal solution. Moreover, because every predefined local region is restricted by a linear classifier, which reduces the flexibility of the classification boundary so as to prevent over-fitting. Simulation results show that our proposed quasi-linear SVM outperforms a standard SVM with linear and nonlinear kernels in the cases of high-noise and high-dimension data sets, where SVM with RBF kernel has overfitting problems.

Bibliography

[1] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[3] S. Menard, *Applied logistic regression analysis*. Sage, 2002, vol. 106.

[4] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[5] V. N. Vapnik, "The nature of statistical learning theory. statistics for engineering and information science," *Springer-Verlag, New York*, 2000.

[6] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[7] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[8] Z. Barutcuoglu, R. E. Schapire, and O. G. Troyanskaya, "Hierarchical multi-label prediction of gene function," *Bioinformatics*, vol. 22, no. 7, pp. 830–836, 2006.

[9] S. Diplaris, G. Tsoumakas, P. A. Mitkas, and I. Vlahavas, "Protein classification with multiple algorithms," in *Advances in Informatics*. Springer, 2005, pp. 448–456.

[10] B. Chen, L. Duan, and J. Hu, "Composite kernel based SVM for hierarchical multi-label gene function classification," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2012, pp. 1–6.

[11] B. Zhou, B. Chen, and J. Hu, "Quasi-linear support vector machine for nonlinear classification," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 97, no. 7, pp. 1587–1594, 2014.

[12] J. Hu, K. Kumamaru, and K. Hirasawa, "A quasi-armax approach to modelling of non-linear systems," *International Journal of Control*, vol. 74, no. 18, pp. 1754–1766, 2001.

[13] B. Chen, F. Sun, and J. Hu, "Local linear multi-SVM method for gene function classification," in *Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on*. IEEE, 2010, pp. 183–188.

[14] H. Zhang, A. C. Berg, M. Maire, and J. Malik, "SVM-KNN: Discriminative nearest neighbor classification for visual category recognition," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. IEEE, 2006, pp. 2126–2136.

[15] H. Cheng, P.-N. Tan, and R. Jin, "Efficient algorithm for localized support vector machine," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 22, no. 4, pp. 537–549, 2010.

[16] Y. Li, Q. Leng, Y. Fu, and H. Li, "Growing construction of conlitron and multiconlitron," *Knowledge-Based Systems*, vol. 65, pp. 12–20, 2014.

[17] W. Li and J. Hu, "Geometric approach of quasi-linear kernel composition for support vector machine," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–7.

[18] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 8, pp. 1798–1828, 2013.

[19] I. Jolliffe, *Principal component analysis*. Wiley Online Library, 2002.

[20] J. Moody and C. J. Darken, “Fast learning in networks of locally-tuned processing units,” *Neural computation*, vol. 1, no. 2, pp. 281–294, 1989.

[21] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*, 1984.

[22] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan kaufmann, 1993, vol. 1.

[23] S. Schulter, C. Leistner, and H. Bischof, “Fast and accurate image upscaling with super-resolution forests,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3791–3799.

[24] D. E. Knuth and A. Raghunathan, “The problem of compatible representatives,” *SIAM Journal on Discrete Mathematics*, vol. 5, no. 3, pp. 422–427, 1992.

[25] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[26] A. Krogh, J. Vedelsby *et al.*, “Neural network ensembles, cross validation, and active learning,” *Advances in neural information processing systems*, vol. 7, pp. 231–238, 1995.

[27] L. Breiman, “Bagging predictors,” *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[28] R. Kohavi, D. H. Wolpert *et al.*, “Bias plus variance decomposition for zero-one loss functions,” in *ICML*, vol. 96, 1996, pp. 275–83.

[29] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[30] A. Asuncion and D. Newman, “Uci machine learning repository,” 2007.

[31] W. H. Wolberg and O. L. Mangasarian, “Multisurface method of pattern separation for medical diagnosis applied to breast cytology.” *Proceedings of the national academy of sciences*, vol. 87, no. 23, pp. 9193–9196, 1990.

[32] P. Horton and K. Nakai, “A probabilistic classification system for predicting the cellular localization sites of proteins.” in *Ismb*, vol. 4, 1996, pp. 109–115.

[33] R. P. Gorman and T. J. Sejnowski, “Analysis of hidden units in a layered network trained to classify sonar targets,” *Neural networks*, vol. 1, no. 1, pp. 75–89, 1988.

[34] H. Güvenir, B. Acar, G. Demiröz *et al.*, “A supervised machine learning algorithm for arrhythmia analysis,” in *Computers in Cardiology 1997*. IEEE, 1997, pp. 433–436.

[35] A. Elisseeff and J. Weston, “A kernel method for multi-labelled classification,” in *Advances in neural information processing systems*, 2001, pp. 681–687.