Kernel-Induced Classification Trees and Random forests

Guangzhe Fan
Department of Statistics and Actuarial Science
University of Waterloo

1. INTRODUCTION

Tree-structured classification models are popular alternatives to classical methods such as logistic regression and discriminant analysis. The CART (Classification and Regression Trees) methodology proposed by Breiman, Friedman, Stone and Olshen (1984) is a very popular and standardized approach in the statistics literature.

CART and most other recursive partitioning algorithms grow a binary decision tree in a sequential fashion by using splitting rules based on predictor variables to split the data in such a way as to reduce variation in a response variable. In CART specifically, the trees are grown as large as possible to avoid early stopping, then pruned backward using a cross-validated cost-complexity criterion to avoid overfitting of the training data and to obtain "right-sized" trees. When the data space is complicated, a single CART model is unstable and may not be able to fulfill a good job. Breiman (1996, 2001) proposed bagging and random forests to improve CART's predictive ability and stability. Basically, the data space is perturbed by either bootstrapped samples or random selections of predictor variables. Then many different CART models are grown on the perturbed data spaces. These different models are used to vote for classifying new cases. Generally, bagging and random forests are more accurate than a single CART model in classification. Some other authors uses linear combination of variables to construct split rules to improve CART and random forests, for example, Loh and Vanichsetakul (1988), Breiman (2001) and Lin and Jeon (2006). There are also research using Markov Chain Monte Carlo stochastic search algorithms to improve CART's construction process, such as Chipman, George and McCulloch (1998). In summary, these variations in CART or random forests essentially use linear-type of split rules on the original input space.

Support vector machine (SVM) is a relatively new but powerful technique in classification. The decision boundary is constructed using only a fraction of the training points, called *support vectors*. As a linear learning machine, it nicely performs a non-linear learning using a variety of *kernel* functions. Detailed descriptions of SVM and *kernel* methods can be found at Burges (1998), Hastie, Tibshirani and Friedman (2001), Scholkopf and Smola (2002), and Shawe-Taylor and Cristtianini (2004). Unlike traditional classification tools such as logistic regression and tree-based models which provide conditional probabilities for classifying an object, SVM only provides a sign of class for the object. Recently, various methods have been proposed to improve SVM in the statistics literature. Zhu and Hastie (2005) designed an import vector machine (IVM) based on *kernel* logistic regression, which is generally equivalent to SVM but has advantages in selecting fewer training points (import points) for computing and in its natural generalization to multicategory classification. Zhu, Su and Chipman (2006) showed that support vector machine is closely related to a radial basis network and could be used in rare-target detection problems. Tang and Zhang (2006) and Liu and Shen (2006) improves SVM on multicategory classification.

This paper mainly aims at improving the tree-based methods such as CART and random forests. It is also connected to SVM via the usage of *kernel* functions. Generally, just as it is not very meaningful to compare logistic regression to CART, it may not be wise either to compare tree-based methods to *kernel*-based methods such as SVM. Each of them could perform well in certain situations and they are themselves based on well developed theories, respectively. In practice, however, we do see that tree-based methods may not perform as well as SVM in some (non-linear) learning situations due to their linear type of split rules in nature. In computation, however, tree-based methods are relatively efficient to implement. For example, there is no need to invert any matrix. Only simple sorting and search with some efficient algorithms is needed in tree construction. Categorical predictor variables and multicategory response variables are also naturally handled by tree-based methods. Conditional probabilities are also estimated by tree-based methods, which could be useful in many applications. More importantly, in the real world, a tree structure may indeed be better to model the data than a linear structure in many cases.

Here we try to take advantages of both tree-based methods and *kernel*-based methods. Specifically, we improve tree-based methods using *kernel* functions while keeping their computational advantages to the most extent. We introduce it as a recursive-partitioned *kernel* learning machine. This is in some sense related to SVM but very different from SVM. First, a recursive partitioning procedure is used; second, the final aim is still to split the data and estimate the conditional probabilities for classification, rather than just to provide a sign of classification.

We observe, from simulated and real world data, that KICT could greatly improve CART and be comparable to random forests and SVM in many situations. As will also be observed, it is still not easy for a simple KICT model to fully win when the space is large and noisy. In these situations, similar to the fact that random forests generally improve

CART, KICT's performance will also be enhanced using the random forests procedure called KIRF: *kernel*-induced random forests.

Before we move to the next section for the methodologies of KICT and KIRF, let's see a motivating example using a well known data set.

Example 1. Breast Cancer Data – Wisconsin

This data set is widely referenced in the literature for classification problems. Its detailed description is at University of California, Irvine Machine Learning Repository: http://www.ics.uci.edu/~mlearn/databases/. The data consist of 683 complete observations on nine predictor variables, with integer values ranging between 1 and 10, and a binary response variable Y ("benign" and "malignant" cases). Let's now pay our attention to one of the observations whose ID is 242970 in the original data, whose attribute values, in the order of the original file, are (5, 7, 7, 1, 5, 8, 3, 4, 1, "benign"). If we only consider the predictor variables vector, then we can write this observation as $\vec{x}^{T}{}_{242970}$ = (5, 7, 7, 1, 5, 8, 3, 4, 1). Now, let's compute the inner products of this observation and every complete observation in the data set,

$$< \vec{x}_{242970}, \quad \vec{x}_{id \in Set_{id}} > = \vec{x}^{T}_{242970} \cdot \vec{x}_{id \in Set_{id}} .$$

We will get 683 inner product values. Further, we can plot these inner product values against their corresponding Y values (0 for "benign", and 1 for "malignant" case).

< insert Figure 1. here>

After some simple sorting and search, we could get a decision line with inner product values =130.5 on the plot, which best classifies the Y variable with only 12 misclassifications on the whole data set. If we claim a split rule as "$\langle \vec{x}_{242970} \cdot \vec{x}_{new} \rangle < 130.5$", we can use it to classify a new observation as *benign* if it is true, and *malignant* otherwise. The resulting decision model is a simple binary tree with two terminal nodes and a training misclassification rate of $12/683 = 1.75\%$. Due to the simplicity of this tree model, it is highly believable that it is going to do as well on future data. Several other observations could be found in the data whose inner product functions could serve as split rules to classify the *Y* variable with similarly small error rates.

From this example, we see that we could take the inner product function based on a specific observation as a new feature in the split rule construction.

2. METHODOLOGY

2.1 *Kernel* Functions

We assume that the learning data set is in standard form as defined in Breiman et al. (1984) with a total of *l* observations. There is a categorical response variable *Y*, which could take a finite set of values. There are *p* candidate predictor variables, X_1, X_2, …, and X_p. Currently, we can assume that they are all quantitative and construct an input vector space X^p. Following the definitions and notations in Shawe-Taylor and Cristtianini (2004), the inner product between observation $\vec{x}_i$ and $\vec{x}_j$ (*i, j =1, 2, 3, … l*), in this real valued vector space X^p, is defined in its standard form,

$$< \vec{x}_i,\ \vec{x}_j > = \sum_{k=1}^{p} x_{ki} \cdot x_{kj}, \tag{1}$$

where, x_{ki} is the real value of variable *k* for observation *i*, *i*=1,2, 3,…, *l*.

Further, let $\boldsymbol{\varphi} :\ X^p \rightarrow F^P$ be a (non-linear) mapping from the input space to an (inner product) feature space, such that

$$\vec{x} = (x_1, x_2, ..., x_p) \mapsto \boldsymbol{\varphi}(\phi_1(\vec{x}), \phi_2(\vec{x}), ...\phi_P(\vec{x}))$$

A *kernel* is a function *K*, such that for all $\vec{x}_i$ and $\vec{x}_j \in X^P$

$$K(\vec{x}_i, \vec{x}_j) = <\phi(\vec{x}_i),\ \ \phi(\vec{x}_j)>. \tag{2}$$

Then (1) can be viewed as a special case of (2) with a trivial mapping. (1) is usually called a linear *kernel*. Many well defined *kernel* functions are functions of the linear *kernels*, making it unnecessary to evaluate the mapped feature vectors explicitly. This is a great advantage of *kernel* methods because the feature space now could be implicitly infinite. Here we give a few examples of *kernels*. A polynomial *kernel* with degree $d \geq 2$ is

$$K(\vec{x}_i, \vec{x}_j) = (<\vec{x}_i,\ \ \vec{x}_j> + c)^d \tag{3}$$

where *c* is a tuning parameter. A Gaussian *kernel* is defined as

$$K(\vec{x}_i, \vec{x}_j) = \exp\{-\|\vec{x}_i - \vec{x}_j\|^2 /(2\cdot\sigma^2)\},$$

$$= \exp\{-(<\vec{x}_i, \vec{x}_i> + <\vec{x}_j, \vec{x}_j> - 2<\vec{x}_i, \vec{x}_j>)/(2\cdot\sigma^2)\} \tag{4}$$

where σ^2 is a tuning parameter. The Gaussian *kernel* is also referred to as a radial basis function (RBF) *kernel*. For simplicity, we omit more descriptions of the properties of *kernels* which are well described in many references.

2.2 KICT: *Kernel*-Induced Classification Trees

From above, we see that each observation *i* in the learning set could interact with other observations via a *kernel* function $K(\vec{x}_i, \cdot)$. The value of this kernel function is uniquely defined by the specified observation vector $\vec{x}_i$ (*i*=1,2,3,…, *l*)and an input observation vector $\vec{z}$ in the space X^p. We could view such an observation-based *kernel* function as a new feature, defined as $K_i(\vec{z}) = K(\vec{x}_i, \vec{z})$. If we have *l* observations in the learning set, then we will have *l* such *kernel*-induced features, $K_i(\cdot)$, *i*=1,2, 3,…, *l*.

Similar to the CART procedure in traditional data space, we could recursively partition the data space using these newly defined *kernel*-induced features and aim to get more homogeneous distributions of the response variable in the resulting terminal nodes of the tree. The tree can grow very large. *V*-fold cross- validation and cost-complexity pruning are used to select the "right –sized" tree. Details of the tree construction technique can be found in Breiman et al. (1984) and Therneau and Atkinson (1997).

For example, setting a *minimum terminal node size* = 5 observations, and using 10-fold cross validation on the breast cancer data, we could obtain a two terminal node tree as mentioned in Example 1. The tree is shown in Figure 2. The *kernel* – induced split rule using a linear *kerne*l function based on the observation of ID 242970 is shown in the root node. The classification of the response variable *Y* (and its distribution: number of class "0" cases / number of class "1" cases) is shown in each of the two terminal nodes.

< insert Figure 2. here>

Computationally, the KICT procedures are relatively efficient because the inner products (or more generally, the *kernel* values) between the training observations need to be computed only once during the search. As to other features of the computing, it is just as efficient as the well-constructed CART and random forests procedure. Categorical predictor variables are not involved in the computation of *kernels*. They can simply be kept as additional features to be used during the KICT construction. Once a KICT is built, a new observation can be classified by computing its related *kernel* function values specified in the split rules recursively from the root node to the terminal node.

2.3 KIRF: *Kernel*-Induced Random forests

KIRF naturally applies the random forests procedure on KICT. The key difference between KIRF and the regular random forests is that during the construction of random trees, instead of selecting a random subgroup of predictor variables, a random subgroup of training observations are selected and their *kernel* functions are used as candidates of split rules. Other procedures and settings of KIRF are generally not different from random forests. From this sense, it retains the computational and theoretical properties of random forests.

2.4 Implementation issues

Kernel functions are very flexible by its definition. It contains all of the information available to the learning machine about the relative positions of the inputs in the feature space (Shawe-Taylor and Cristianini 2004, page 70). Which type of *kernels* to choose depend on the understanding of the problems and more importantly, on the problems themselves. Currently we limit our use to a few basic types of kernels such as linear and Gaussian in this introductory paper. In practice, whether to use KICT or KIRF as alternatives to CART, random forests or SVM for a given problem, or specifically which *kernel* to use can generally be determined by cross-validation.

3. EXAMPLES AND RESULTS

Here we will use simulated data and well known benchmark data to illustrate the performances of KICT and KIRF in contrast to CART, random forests and SVM. All implementations are done in R using standard libraries. We use RPART (Therneau and Atkinson 1997) as an implementation of CART. For KICT and RPART, a minimum node size of 5 observations is imposed and 10-fold cross-validation is used to select the tree with the smallest cross-validated error (we are not using the 1-SE rule since the tree size is not of major interest here and the classification accuracy is usually slightly worse if using 1-SE rule in practice). For random forests, the default settings in the R library are used. KIRF follows the same settings as random forests in all situations.

We consider two basic types of kernels: the linear *kernel* and the Gaussian *kernel* for simplicity of illustration and comparison. Since a tree algorithm is invariant to monotonic transformation of data, the Gaussian *kernel* parameter σ^2 can be simply set to be 1.0 for KICT. For SVM, tuning is not an easy task. Here we follow the ways adopted by published papers, such as Zhu and Hastie (2005), to use Gaussian *kernel* with the σ^2 values searched over a range of possible values to get its best generalization error. Our results of SVM on some well known data sets generally match previously published results of fine-tuned SVM, such as those in Zhu and Hasie (2005).

3.1 A simulated example

This example is used to illustrate how KICT and KIRF work and how they are different from CART (RPART) and random forests, respectively. We have two variables X_1 and X_2 uniformly and independently distributed on interval [-1, 1]. The binary response variable *Y* is determined by X_1 and X_2 using the following equation.

$$Y = \begin{bmatrix} 0, & \sqrt{X_1^2 + X_2^2} < 0.3 \\ 1, & 0.3 \le \sqrt{X_1^2 + X_2^2} < 0.6 \\ 0, & 0.6 \le \sqrt{X_1^2 + X_2^2} < 0.9 \\ 1, & 0.9 \le \sqrt{X_1^2 + X_2^2} < 1.2 \\ 0, & \mathit{otherwise} \end{bmatrix}$$

We simulate 4200 observations and the first 200 observations are used as training set to construct models. Then the trained models are used to classify the 4000 unused observations. Certainly, the Gaussian *kernel* is a good choice for KICT here. The experiment is done with 20 realizations using different random seeds and the results are summarized in Table 1. For illustration purpose, we plot the KICT tree for one random realization in Figure 3. The split variables are the Gaussian *kernel* functions based on the specified observations: No. 193, No. 88, No. 115, No. 155 and No. 83 of the training set, respectively. We also plot the test set and the fitted results of the test set for each method in Figure 4. The summarized results of 20 random realizations are shown in Table 1. We see that the KICT trees are generally small in size but it does well in discovering the relationship between *Y* and the predictor variables. SVM performs similarly well, which is expected for this type of problems. RPART and random forests use linear type of split rules and do not perform well for this problem, although random forests do improve RPART quite a lot. Similarly, KIRF improves KICT and performs the best among the methods not only in terms of classification accuracy, but also in its stability over the random replications.

<insert Figure 3. here>

<insert Figure 4. here>

<insert Table 1. here>

3.2 Benchmark Data Sets

To further explore the potential of KICT and KIRF in classification, we try some benchmark data sets. Most of them are available in the UCI machine learning repository. The *Kyphosis* and *Iris* data are also attached in R. For the real data sets, we randomly split the data into training and test set with 90% and 10% of the observations, respectively, and repeat 100 times to get the average result. The *Kyphosis* data set has only 81 observations so a leave-one-out cross-validation is actually used. The *Ringnorm*, T*wonorm* and *Waveform* data are generated using the "mlbench" library in R and the experiment is repeated 20 times. For some real data sets, their predictor variables are scaled. Scaling is a default choice for the SVM method in R and it also improves KICT and KIRF generally, especially when the scales of the predictor variables are very different. Certainly, RPART and random forests are invariant to scaling.

The detailed descriptions of the data sets are in Table 2. The average test set misclassification rates and the model sizes with the standard errors in the parentheses are shown in Table 3. From the results we can see that KICT significantly improves CART (RPART). In some data sets, such as *Ringnorm* and *Twonorm*, the misclassification rate reduces over 80%. Its results are also comparable to those of random forests and SVM. KIRF further improves KICT in most situations and performs mostly the best of all methods. There are, however, some small variations in the results. For example, for the two real data sets, *Breast cancer* and *Iris*, the KICT models are numerically slightly better than the KIRF models. Generally, and especially for the synthetic data sets, KIRF improves KICT significantly.

< insert Table 2. here>

< insert Table 3. here>

4. SUMMARY and DISCUSSIONS

We introduce the concept of constructing classification trees using *kernel* functions (KICT). Further, the ensemble method is applied to KICT and further improves it, namely, KIRF. Both methods show strong classification ability in many examples. Certainly, KICT is not as easy to interpret as CART. This could be a future direction of research.

As mentioned in the introduction, there are other variations of CART and random forests, for example, Breiman (2001) and Lin and Jeon (2006), which are mainly based on linear combination of predictor variables. Based on the benchmark data sets, we can compare our results to their previously published results and again show our advantages of classification accuracy. Certainly, KIRF itself can be further improved using these previous published ideas, such as using a linear combination of *kernel* functions to construct split rules. This could be a direction of future research.

In summary, due to the popularity of CART serving as basis of a variety of learning methods, and due to the richness and flexibility of *kernel* functions in a variety of applications, KICT and KIRF would be useful classification tools in many problems.

REFERENCES

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984), *Classification and Regression Trees*, Belmont, CA: Wadsworth.

Breiman, L. (1996), "Bagging Predictors," *Machine Learning*, 24, 123-140.

— (2001), "Random forests," *Machine Learning*. 45, 5-32.

Burges, C. (1998), "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery*, 2, 121-167, Boston: Kluwer.

Chipman, H., George, E., and McCulloch, R. (1998), "Bayesian CART Model Search (with discussions)," *Journal of the American Statistical Association*, 93, 935-960.

Hastie, T., Tibshirani, R., and Friedman, J. (2001), *The Elements of Statistical Learning, Data Mining, Inference and Prediction*, New York: Springer-Verlag.

Lin, Y. and Jeon, Y., (2006), "Random forests and Adaptive Nearest Neighbors," *Journal of the American Statistical Association,* 101, 578-590.

Liu, Y., and Shen, X. (2006), "Multicategory ψ - Learning," *Journal of the American Statistical Association,* 101, 500-509.

Loh, W., and Vanichsetakul, N. (1988), "Tree-structured Classification Via Generalized Discriminant Analysis," *Journal of the American Statistical Association*, 83, 715-725.

Shawe-Taylor, J., and Cristianini, N. (2004), *Kernel Methods for Pattern Analysis*, Cambridge, UK, Cambridge University Press.

Scholkopf , B., and Smola, A. (2002), *Learning with Kernels – Support Vector Machines, Regularization, Optimization, and Beyond*, Cambridge, Massachusetts, The MIT Press.

Tang, Y., and Zhang, H. (2006), "Multiclass Proximal Support Vector Machines," *Journal of Computational and Graphical Statistics*, 15, 339-355.

Therneau, TM, and Atkinson, EJ. (1997), "An introduction to recursive partitioning using the RPART routines," Technical report, Mayo Foundation.

Zhu, J., and Hastie, T. (2005), "Kernel Logistic Regression and the Import Vector Machines," *Journal of Computational and Graphical Statistics*, 14, 185-205.

Zhu, M., Su, W., and Chipman, H. (2006), "LAGO: A Computationally Efficient Approach for Statistical Detection," *Technometrics*, 48, 193- 205.

Table 1. Summarized results of Simulation 3.1. The average performances (standard errors) over 20 random realizations are shown.

Misclassification Rate % (standard error %)						
KICT	KIRF	RPART	Random forests	SVM	KICT size	RPART size
13.45 (3.35)	9.13 (1.87)	30.95 (3.93)	21.78 (3.15)	13.36 (2.44)	6.80 (1.36)	18.25 (2.92)

Table 2. Summary of benchmark data sets. *l* is the training set size, *N* is the test set size, and *p* is the number of predictor variables.

Data	l	N	p	No. of classes	No. of replications	Scaling
Breast Cancer	613	70	9	2	100	No
Iris	135	15	4	3	100	Yes
Kyphosis	80	1	3	2	81	Yes
Thyroid	194	21	5	3	100	Yes
Ringnorm	400	7000	20	2	20	No
Twonorm	400	7000	20	2	20	No
Waveform	400	7000	21	3	20	No

Table 3. Summary of classification results for the benchmark data sets.

Data	Choice of *kernel*	Misclassification rate % (Standard error %) KICT	KIRF	RPART	Random Forest	SVM	KICT size	RPART size
Breast Cancer	*linear*	2.21 (1.75)	3.11 (1.82)	5.39 (2.88)	2.79 (1.75)	3.00 (1.75)	2 (0)	7.18 (2.12)
Iris	*Gaussian*	4.40 (4.85)	4.73 (4.67)	6.53 (5.99)	5.13 (5.67)	6.60 (8.76)	3.05 (0.22)	3.49 (0.54)
Kyphosis	*Gaussian*	23.46 (42.64)	17.28 (38.05)	24.69 (43.39)	23.46 (42.64)	22.22 (41.83)	2.20 (1.18)	1.30 (1.17)
Thyroid	*linear*	4.28 (3.28)	2.67 (3.41)	7.86 (6.92)	4.05 (4.71)	4.24 (4.38)	3.00 (0.00)	4.42 (0.92)
Ringnorm	*Gaussian*	3.50 (.51)	1.61 (.16)	21.06 (2.10)	6.27 (.89)	1.78 (.15)	2.75 (.85)	12.75 (3.09)
Twonorm	*linear*	5.64 (.89)	2.51 (.16)	22.78 (.90)	3.72 (.36)	2.52 (.19)	3.55 (1.23)	14.4 (2.39)
Waveform	*Gaussian*	20.46 (1.17)	15.09 (.65)	28.33 (2.46)	16.04 (.53)	15.12 (.48)	9.85 (3.51)	14.35 (6.66)

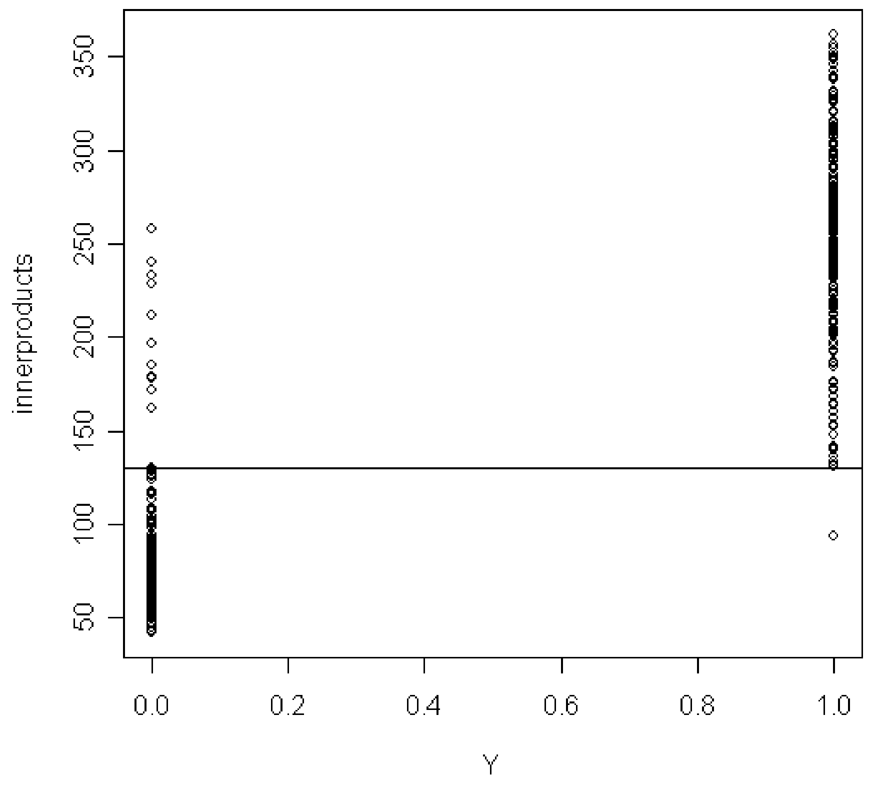


Figure 1. Plot of inner product values between the specified observation (*ID* 242970) and all observations in the data against the binary response variable *Y*.

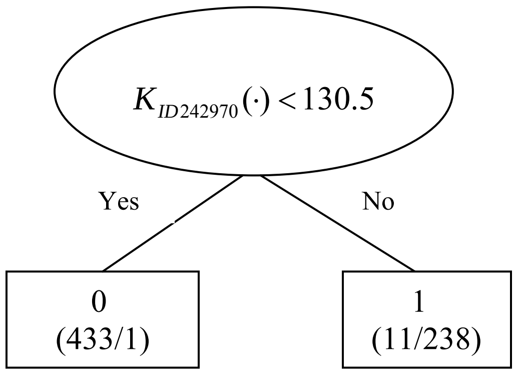


Figure 2. *Kernel*-induced classification tree model for the Breast Cancer data.

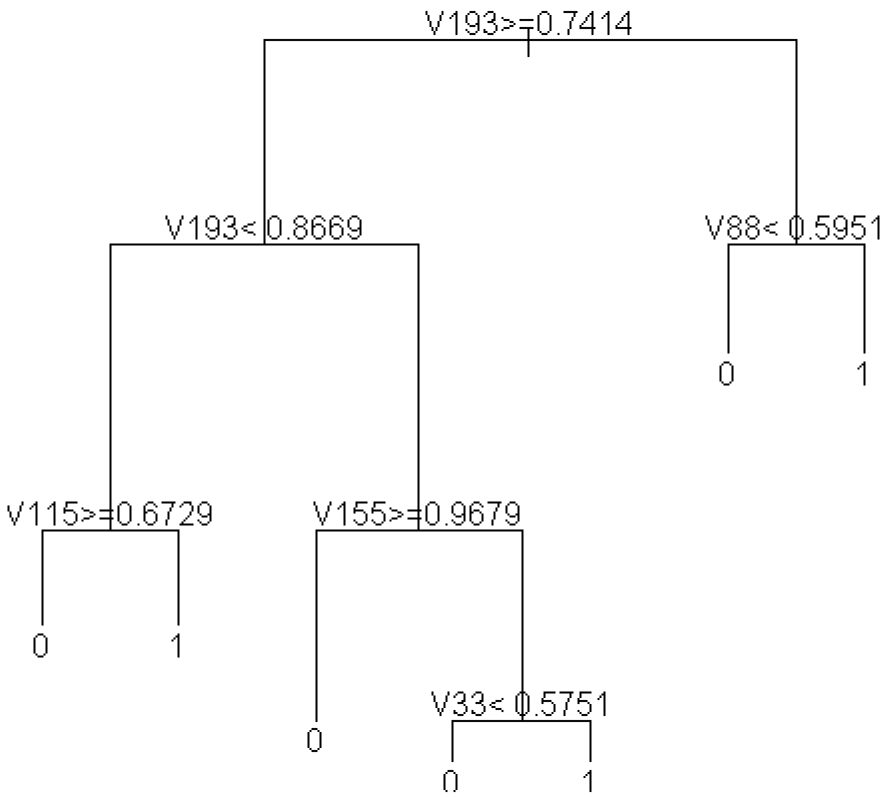
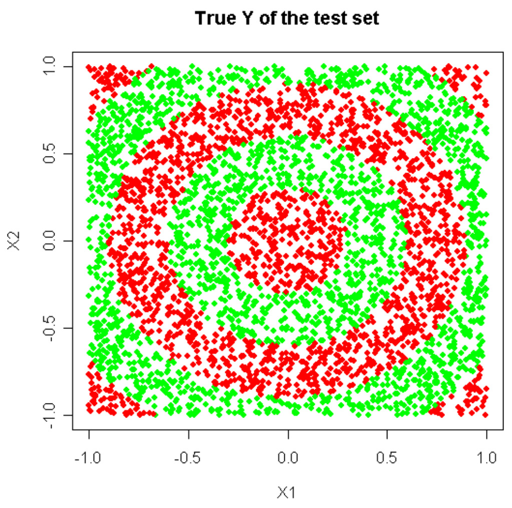
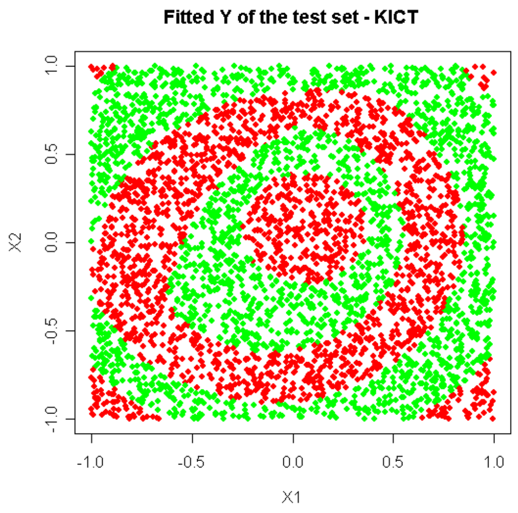
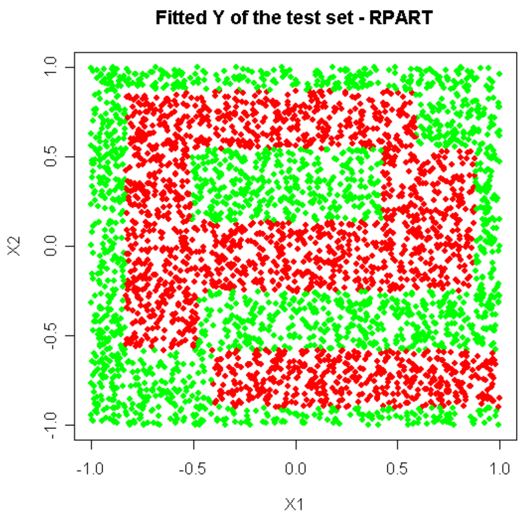
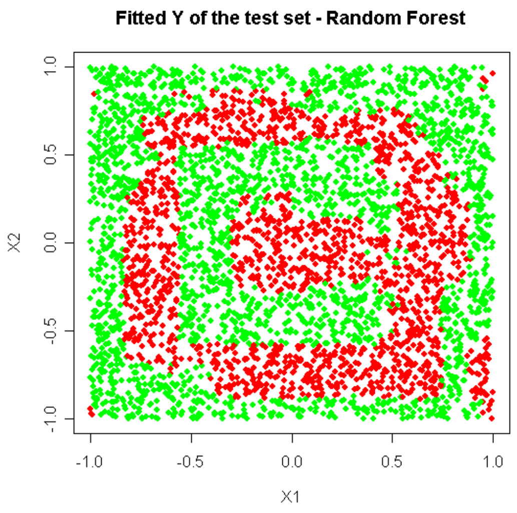


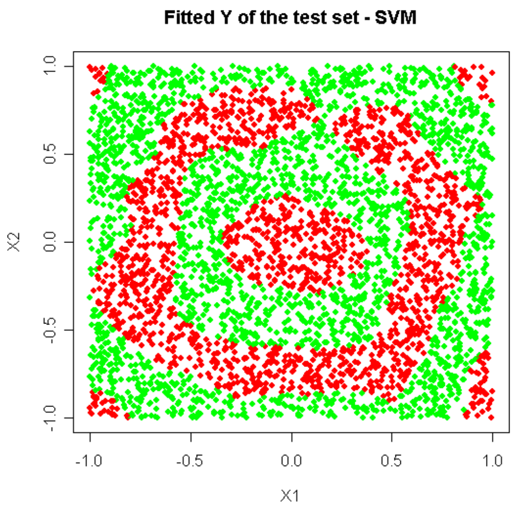
Figure 3. The seven-terminal nodes KICT model for one random realization of the simulated data.


True Y of the test set
X2
X1
1.0
0.5
0.0
-0.5
-1.0


Fitted Y of the test set - KICT
X2
X1
1.0
0.5
0.0
-0.5
-1.0


Fitted Y of the test set - RPART
X2
X1
1.0
0.5
0.0
-0.5
-1.0


Fitted Y of the test set - Random Forest
X2
X1
1.0
0.5
0.0
-0.5
-1.0


Fitted Y of the test set - SVM
X2
X1
1.0
0.5
0.0
-0.5
-1.0

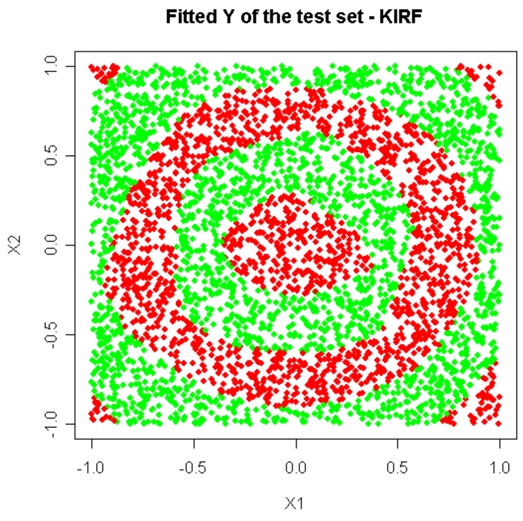

Fitted Y of the test set - KIRF
X2
X1
1.0
0.5
0.0
-0.5
-1.0

Figure 4. Plot of the true *Y* values and the fitted *Y* values of different models for the test set in one random realization of Simulation 3.1. The dark (red) points denote class “0” of *Y*.