Diabetes Classification

Group 4: Crystal Dong, Juan Du, Yanxing Zhao, Zhenhuan Cui, Lynn Friedman

I. Introduction

Literature

The problem we will deal with is to determine whether a woman has diabetes given knowledge of eight possible explanatory variables. The population is Pima Native American women living near Phoenix, Arizona, USA.

Smith, J. W., Everhart, J. E., Dickson, W. C., Knowler, W. C., Johannes, R. S. (1988) used the ADAP learning algorithm to forecast the onset of diabetes mellitus. The diagnostic, binary-valued variable investigated was whether the patient shows signs of diabetes according to World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose was at least 200 mg/dl at any survey or routine medical examination). Their ADAP algorithm made a real-valued prediction between 0 and 1, which was then transformed to a binary decision using a cutoff value of 0.448. Using 576 training instances, the sensitivity and specificity of their algorithm was 76% on the remaining 192 instances. ADAP is an adaptive learning routine that generates and executes digital analogs of perceptron-like devices. It is a unique algorithm.

W. Zhou (http://www.prettyview.com/ann/diabetes.shtml) used a neural network model. He randomly selected 170 observations each from the classes of those with diabetes and those without to represent the training data. The remaining 428 instances were left as testing observations. Every feature selection process started with eight input nodes, each corresponding to one variable, and two output nodes, corresponding to the two possible test results. Each training session was run for 500 cycles. The process was carried out a total of five times. After each iteration, one input was deleted, and the average MCSR and average CASR[1] were calculated. Zhou found that as inputs were deleted one after another, the MCSR's and CASR's reversed the trend of decline when four and three inputs, respectively, were kept. In particular, the highest MCSR's were reached when four inputs were used. In other words, to achieve the highest success rate for both "positive" and "negative" cases, only four inputs were needed: 1) number of times pregnant, 2) plasma glucose concentration at 2 hours in an oral glucose tolerance test, 3) body mass index and 4) diabetes pedigree function. The success rate was above 50% even with only one input, plasma glucose concentration at 2 hours in an oral glucose tolerance test, was used.

The dataset was also studied extensively in Ripley, B.D. (1996) *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press). Most of their illustrations omitted serum insulin because of 376 missing values and used the 532 complete records on the remaining variables. These were randomly split into a training set of size 200 and a test set of size 332. Methods which can deal with missing values were given 100 of the incomplete cases as part of the training set.

---

[1]MCSR: The **M**inimum **C**lass **S**uccess **R**ate was the lowest success rate among all the target classes. The average MCSR is the MCSR's averaged over the five[1] training sessions within each process.
CASR: The **C**lass **A**verage **S**uccess **R**ate is the success rate averaged over all the target classes. The average CASR is the CASR's averaged over the five training sessions within each process.

In *Pattern Recognition and Neural Networks*, the best methods reduced the error rate to about 20%. Standard linear discrimination made 67/332 errors on the test set and choosing a subset of variables by cross-validation on the training set suggested no reduction was worthwhile. Most errors were made on the group reported to have diabetes(42/109), where quadratic discrimination did significantly worse(84/109). Logistic regression dropped blood pressure and skin thickness by a stepwise selection procedure and made 66/332 test errors. With nearest neighbor 10-fold cross validation, the error on training sets were 57, 65, 57, 54, 51 and 55(out of 200) for k = 1, 3, 5, 7, 9, 11; for leave-one-out they obtained 58, 65, ,60, 55, 49 and 41. On the test set the numbers of errors were 98 for k = 1 and 82 for k = 9, out of 332. Test set error was about 70/332 using OLVQ1 plus LVQ2 or LVQ3 with 2-4 vectors per class. Fitting MARS and PPR models by least squares did not improve the fit over linear methods, with a typical test-set error rate of 75/332. Classification tree with pruning resulted a test error of 81/332, while using an additional 100 partially missing examples made 74/332 errors on the test set.

The following table summarizes the results of some past studies of the data set.

	Error	Rate		Error	Rate		Error	Rate
Algorithm	Train	Test	Algorithm	Train	Test	Algorithm	Train	Test
LogDisc	0.219	0.223	Bayes	0.239	0.262	IndCart	0.079	0.271
Dipol92	0.22	0.224	C4.5	0.131	0.27	BayTree	0.008	0.271
Discrim	0.22	0.225	BackProp	0.198	0.248	LVQ	0.101	0.272
Smart	0.177	0.232	Cal5	0.232	0.25	Kohonen	0.134	0.273
Radial	0.218	0.243	Cart	0.227	0.255	Ac2	0	0.276
Itrule	0.223	0.245	Castle	0.26	0.258	Newld	0	0.289
Alloc80	0.288	0.301	QuaDisc	0.237	0.262	Cn2	0.01	0.289
KNN	0	0.324	Default	0.35	0.35			

(Source: http://www.pmsi.fr/diabind0.htm)

We use the nearest neighbor method to impute missing values. Logistic regression is used as preliminary classification study. More attention is focused on classification tree and support vector machin. We divide the dataset into a training set of size 200 and a test set of size 568 to compare the predictability of different methods.

II. Preliminary Analysis

The data can be found at http://ftp.ics.uci.edu/pub/machine-learning-databases/pima-indians-diabetes.

The original owner of the dataset is National Institute of Diabetes and Digestive and Kidney Diseases. It was donated by Vincent Sigillito (vgs@aplcen.apl.jhu.edu) Research Center, RMI Group Leader Applied Physics Laboratory at Johns Hopkins University. The date that the dataset was received by UCI is 9 May 1990. It was noted that several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

There are 768 instances to be studies. The dataset has eight attributes (explanatory variables) and one class variable. All eight attributes are numeric-valued. They are

V1: “npreg”—Number of times pregnant
V2. “gluc”—Plasma glucose concentration a 2 hours in an oral glucose tolerance test
V3. “bp”—Diastolic blood pressure (mm Hg)
V4. “tric”—Triceps skin fold thickness (mm)
V5. “insu”—2-Hour serum insulin (mu U/ml)
V6. “bmi”—Body mass index (weight in kg/(height in m)^2)

V7. “ped”—Diabetes pedigree function
V8. “age”—Age (years).

The class variable takes two values. The value 1 indicates a test of positive for diabetes while 0 indicates negative. There are 268 positive cases and 500 negative cases in the dataset.

The observations in this dataset were selected from a larger database. Among the constraints are that all patients here are females of at least 21 years of age of Pima Indian heritage.

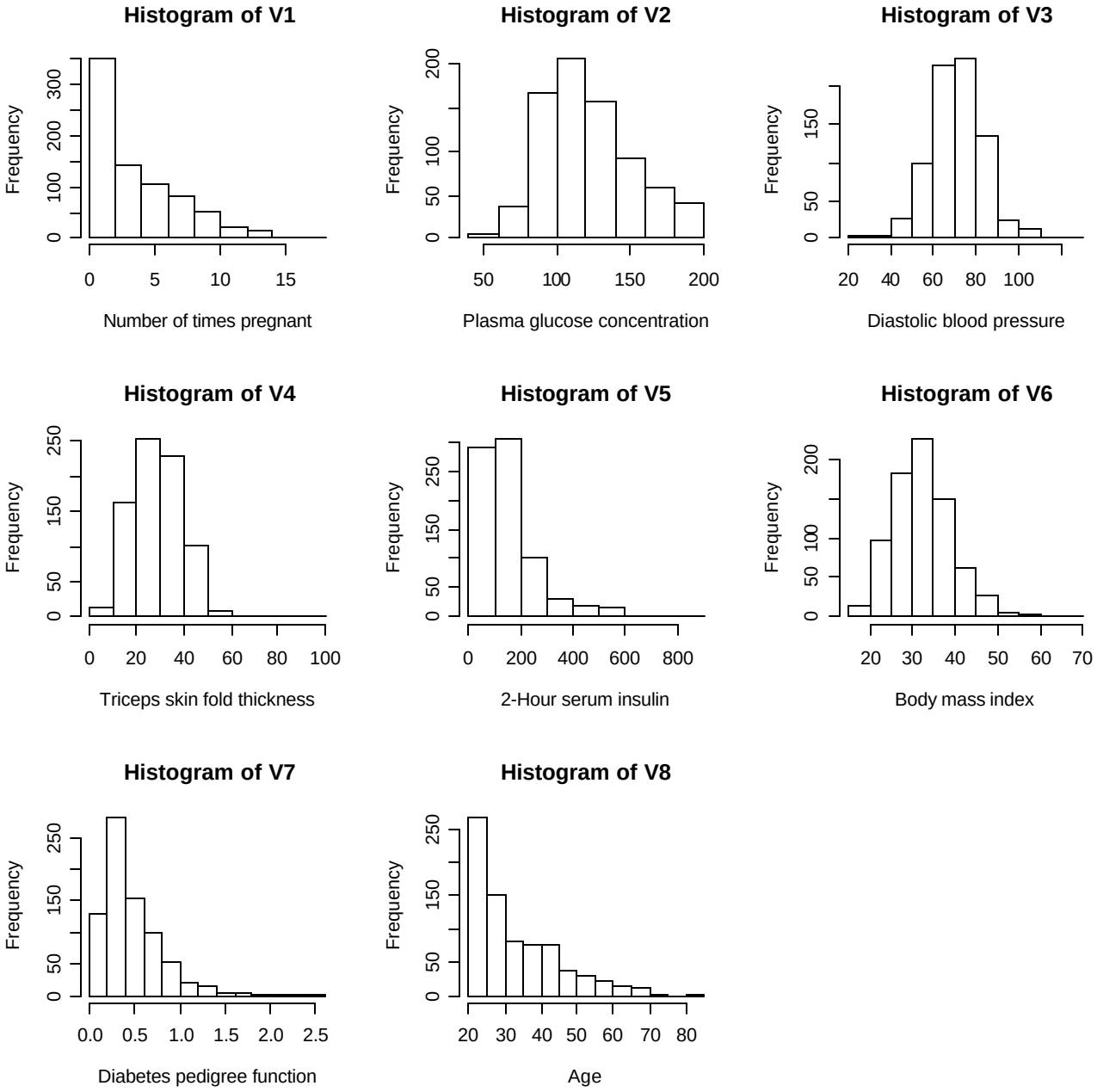
A brief summary of attributes are

	Attribute:	Mean:	St. Dev.:
1	“npreg”	3.8	3.4
2	“gluc”	120.9	32
3	“bp”	69.1	19.4
4	“tric”	20.5	16
5	“insu”	79.8	115.2
6	“bmi”	32	7.9
7	“ped”	0.5	0.3
8	“age”	33.2	11.8

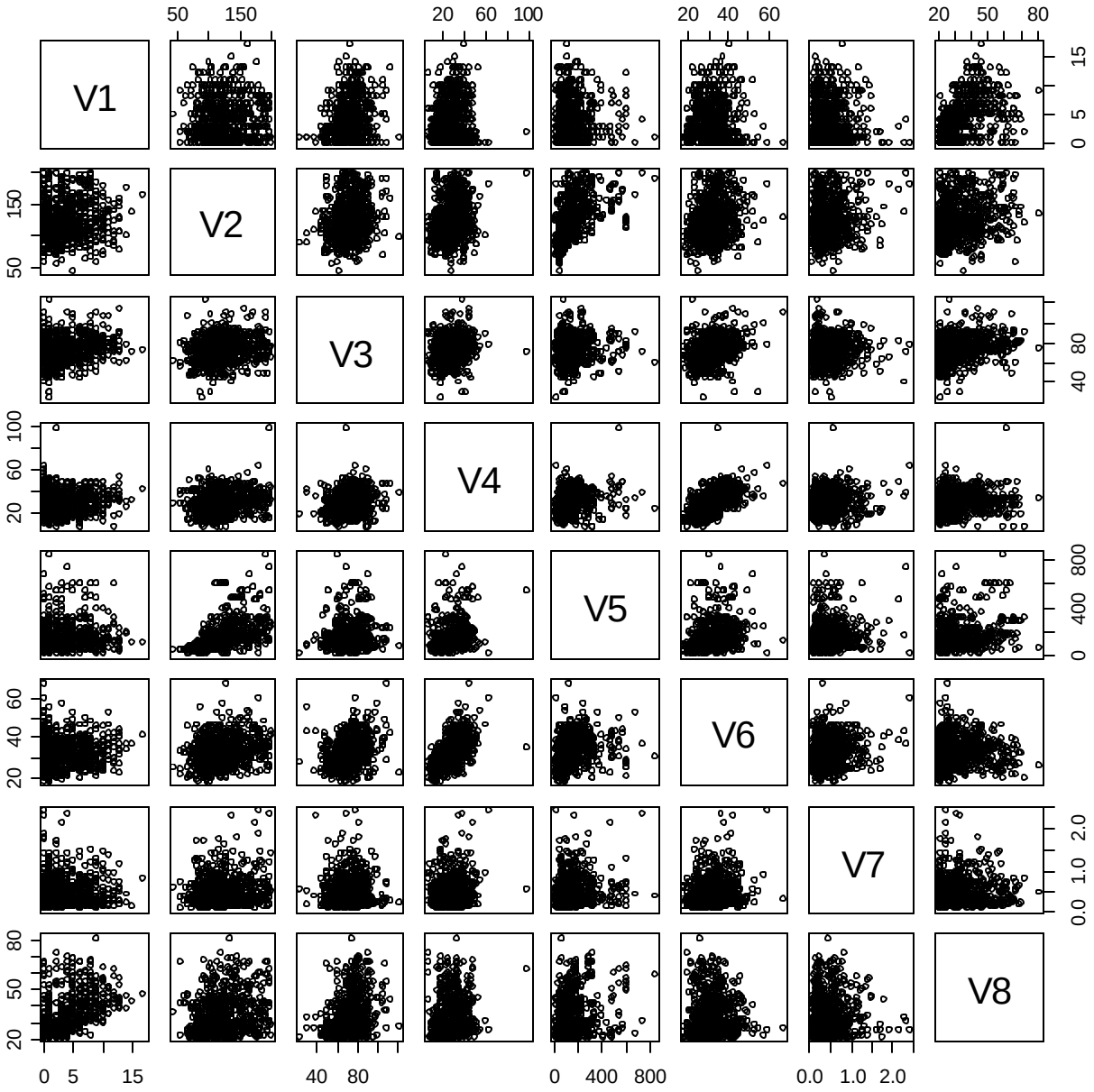
The description of these data on the “names” file indicates that there are no missing data. However, there apparently are missing data for plasma glucose concentration, diastolic blood pressure, triceps skin fold sickness, 2-hour serum insulin, and body mass index. Any zero values for these must be the result of lack of measurement, and there are many 0’s in these data.

variable	no. missing	% missing
glu	6	0.007813
bp	35	0.045573
skin	227	0.295573
insu	374	0.486979
bmi	11	0.014323

The following method was used to correct the problem: each missing datum was replaced with the corresponding value of the nearest neighbor which had no missing values. Euclidian distance was used to compute the distance between two data points. Here are the histograms for the input variables after imputation.



The imputation method worked well as shown in the following pair-wise scatter plots of the eight explanatory variables after imputation. Notice that the preservation of intuitive correlations between variables; for example, body mass index and triceps skin fold thickness.



And here is the summary statistics for the imputed explanatory variables.

Descriptive Statistics:

Variable	N	Mean	Median	TrMean	StDev	SE Mean
Pragnent	768	3.845	3.000	3.604	3.370	0.122
Plasma G	768	121.78	117.00	120.80	30.58	1.10
Diastoli	768	72.280	72.000	72.182	12.393	0.447
Triceps	768	29.242	29.000	29.048	10.402	0.375

2-Hour S	768	155.68	123.50	141.82	116.56	4.21
Body Mas	768	32.442	32.150	32.206	6.893	0.249
Diabetes	768	0.4719	0.3725	0.4375	0.3313	0.0120
Age	768	33.241	29.000	32.233	11.760	0.424

Variable	Minimum	Maximum	Q1	Q3
Pragnent	0.000	17.000	1.000	6.000
Plasma G	44.00	199.00	99.00	141.00
Diastoli	24.000	122.000	64.000	80.000
Triceps	7.000	99.000	22.000	36.000
2-Hour S	14.00	846.00	77.00	190.00
Body Mas	18.200	67.100	27.500	36.600
Diabetes	0.0780	2.4200	0.2433	0.6268
Age	21.000	81.000	24.000	41.000

III. Main Analysis

Method Overview

Classification Methods Overview

(i) Classification Tree

The strategy for building a tree is to grow a large tree until some minimum node size (the number of observations in each node) is reached, or some statistical criterion is satisfied. Starting with all of the data, consider an input variable j and split point s, and define the pair of half-planes $R_1(j,s)=\{X \mid X_j \le s\}$ and $R_2(j,s)=\{X \mid X_j > s\}$. We want to find the j and s that minimize the sum of a loss function in each of the half-planes. After finding the best variable, and splitting our data with regard to this variable, we take the observations in these splits and look for next best variables, etc. until we reach and have the data split into rectangular regions.

For k = 1, 2, and N_m = number of observations in each class, let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_m = k),$$

the proportion of class 1 or 2 in each node m. We classify the observations in node m to class k(m) the majority class in node m. The measure of impurity is called *entropy* or *information*:

$$\sum_{j \neq k} p_{mj} p_{mk} = 1 - \sum_{k} p_{mk}^2$$

Where p_{ik}^2 can be estimated by $\hat{p}_{mk}$. The tree construction continues until the number of cases reaching each leaf is less than 10 or the leaf has less than 1% entropy than the root.

Then the tree is pruned using cost-complexity pruning. If we let R_i denote the the entropy evaluated at the leaves, and let R denote the entropy of the tree. The size of the tree is the number of leaves. The complexity measure is then:

$$R_a = R + a\,size$$

Breiman et al. showed that the set of rooted subtree which minimiz the cost-complexity measure is nested. Thus we can find the optimal trees by a sequence of snip operations on the current tree as we increase the a . We use 10 time cross validation to choose the degree of pruning. We can compute the entropy versus a for the pruned trees. We look for the a value that minimizes the deviance.

We used the recursive partitioning and regression trees (rpart) function from the R library to obtain classification trees for each of the training/testing sets. The algorithms automatically decide on the splitting variables and split points and also what topology the tree should have. We used "printcp" and "prune" to print the cp table and prune the tree according to the best cp. The "summary.rpart()" command can direct the output tree to a file. The training error and testing error were calculated as the misclassification rate.

(ii) Logistic Regression

Since the response variable is whether the patient showed signs of diabetes or not, we can fit a logistic regression model to this data set. In our study, the probability of showing signs of diabetes is modeled by

$$p(x) = \frac{e^{b_0 + \sum_{i=1}^{k} b_i x_i}}{1 + e^{b_0 + \sum_{i=1}^{k} b_i x_i}}$$

The link function is the logit:

$$g(x) = \ln\left\{\frac{p(x)}{1-p(x)}\right\} = b_0 + \sum_{i=1}^{k} b_i x_i$$

In the final model, the coefficient for a variable is the log odds ratio of showing signs of diabetes if increase the corresponding variable by one unit. The estimation of parameters is done using maximum likelihood. To assess the model fit, we can group the data by percentiles of the estimated probabilities and use the chi-squared test. To decide the cutoff points for classification into diabetic vs. non-diabetic,

we can plot the sensitivity vs (1-specificity) and obtain the ROC curve. The area under the ROC curve is a measure of discrimination. An ROC>=0.7 is considered acceptable discrimination.

The command "glm" with "family=binomial" was used in S-plus when the model was fit. Model selection was based on the AIC criterion, which uses the log likelihood (see p. 204, text).

(iii) Support Vector Machine

Our training data consists of N pairs $(x_1, y_1), (x_2, y_2), \ldots\ldots (x_N, y_N)$, with $x_i \in \Re^p$ and $y_i \in \{-1,1\}$. Define a hyperplane by

$$\{x : f(x) = x^T b + b_0\},$$

where b is a unit vector: $\| b \|$=1. A classification rule induced by f(x) is

$$G(x) = sign[x^T b + b_0]$$

For separable case, the hyperplane that creates the biggest margin between the training points for class -1 and 1 is the following optimization problem:

$$\max_{b, b_0, \| b \| = 1} C \text{ subject to } y_i (x_i^T b + b_0) \geq C, i = 1, \ldots\ldots, N,$$

which is equivalent to

$$\min_{b, b_0} \| b \| \text{ subject to } y_i (x_i^T b + b_0) \geq 1, i = 1, \ldots\ldots, N$$

For non-separable case, we still maximize ||C||, but allow for some points to be on the wrong side of the margin. Define the slack variables $x = (x_1, x_2, \ldots\ldots, x_N)$. The support vector classifier is:

$$\min \| b \| \text{ subject to } y_i (x_i^T b + b_0) \geq 1 - x_i \forall i, x_i \geq 0, \sum x_i \leq \text{constant}$$

The problem is quadratic with linear inequality constraints, hence it is a convex optimization problem. We describe a quadratic solution using Lagrange multipliers. Computationally, it is convenient to be re-expressed in the equivalent form

$$\min \frac{1}{2} \| b \| + g \sum_{i=1}^{N} x_i$$

subject to $y_i(x_i^T b + b_0) \geq 1 - x_i \forall i, x_i \geq 0$

which is equivalent to the optimization problem

$$\min_{b, b_0} \sum_{i=1}^{N} [1 - y_i f(x_i)]_+ + l \parallel b \parallel^2$$

with $l = 1/(2g)$.

(iv) Relationship between logistic regression and support vector machine

From optimization point of view, both logistic regression and support vector machine are the minimizing functions of some loss functions.

Loss Function	L(Y, F(X))	Minimizing Function
(-)Binomial Log-likelihood	$\log(1 + e^{Yf(X)})$	$f(X) = \log \frac{P(Y = +1 \mid X)}{P(Y = -1 \mid X)}$
Support Vector Machine	$[1 + Yf(X)]_+$	$f(X) = \begin{cases} +1, & if \quad P(Y = +1 \mid X) \geq \frac{1}{2} \\ -1, & Otherwise \end{cases}$

Linear logistic regression uses $\hat{f}(x) = x^T b + b_0$ to approximate

$$f(X) = \log \frac{P(Y = +1 \mid X)}{P(Y = -1 \mid X)},$$

while linear support vector machine uses $\hat{G}(x)$=sign[$x^T b + b_0$] to estimate

$$f(X) = \begin{cases} +1, & if \quad P(Y = +1 \mid X) \geq \frac{1}{2} \\ -1, & Otherwise \end{cases}$$

This casts the SVM as a regularized function estimation problem, where the coefficients of the linear expansion $\hat{f}(x) = x^T b + b_0$ are shrunk toward zero.

Given that the observations in the training set come from the same distribution as the target population and the losses of misclassification into each class are the same. This is called the standard the situation. If we use 0-1 loss for misclassification, the Bayes rule minimizing the expected loss is:

$$f_B(x) = \begin{cases} +1, & if \quad \dfrac{p(Y=+1 \mid X)}{1-p(Y=+1 \mid X)} > 1 \\ -1, & Otherwise \end{cases}$$

In logistic regression, we use MLE, which is consistent and asymptotically efficient. If we choose logit=0 or p(x)=0.5 as the cut-off point of the classification, it is approximating the Bayes rule.

According to Lin(1999), if the reproducing kernel Hilbert space is rich enough, the solution to the non-linear support vector machine approaches the Bayes rule as the sample size tends to infinity. Hence, we can not guarantee that linear support vector machine (use linear kernel) is a good approximation of the Bayes rule under the standard situation.

If the sample distribution differs from the that of the target population and the cost of misclassification into different classes are not equal, the Bayes rule minimizing the expected loss becomes:

$$f_B(x) = \begin{cases} +1, & if \quad \dfrac{p(Y=+1 \mid X)}{1-p(Y=+1 \mid X)} > \dfrac{c^+}{c^-}\dfrac{p_s^+}{p_s^-}\dfrac{p^-}{p^+} \\ -1, & Otherwise \end{cases}$$

where p_s^+, p_s^- denote the proportion of the corresponding class in the training sample; p^+, p^- denote the proportion of the corresponding class in the target population; c^+, c^- denote the cost for false positive and false negative.

In this scenario, all we should do to logistic regression is to simply change the threshold of the classification problem and all the conclusions about logistic regression hold in this situation. While for support vector machine, we have to modify the loss function and redo the quadratic programming procedure to obtain the estimate of the minimizing function. In this non-standard situation, the loss function is changed to:

$$\frac{1}{N}\sum_{i=1}^{N} L(y_i)[(1-y_i f(x_i))_+]$$

where $L(-1) = c^+ p_s^+ p^-$ and $L(1) = c^- p_s^- p^+$. The penalty term remains unchanged. The solution to this regularization problem also tends to the Bayes rule under certain conditions (see Lin (2002) Support Vector Machines for Classification in Nonstandard Situation).

The reason why we have to go through all the steps for SVM is that SVM does not give the whole picture of probability and we loss most of the information when the estimated boundary approaches the step function of the Bayes rule. [For linear support vector machine we can retrieve the probability under some conditions (See John Platt (1999) Probalistic Outputs for Support vector machines and comparison to Regularized Likelihood Method)]

Implementation:

There are totally 768 observations. We randomly select 200 observations for the training set and put the remaining 568 observations into the testing set. We repeat the random split 10 times and implement the classification on each split for training error and testing error. Finally we calculate the mean and std. for those errors.

Method 1: Classification Tree

1

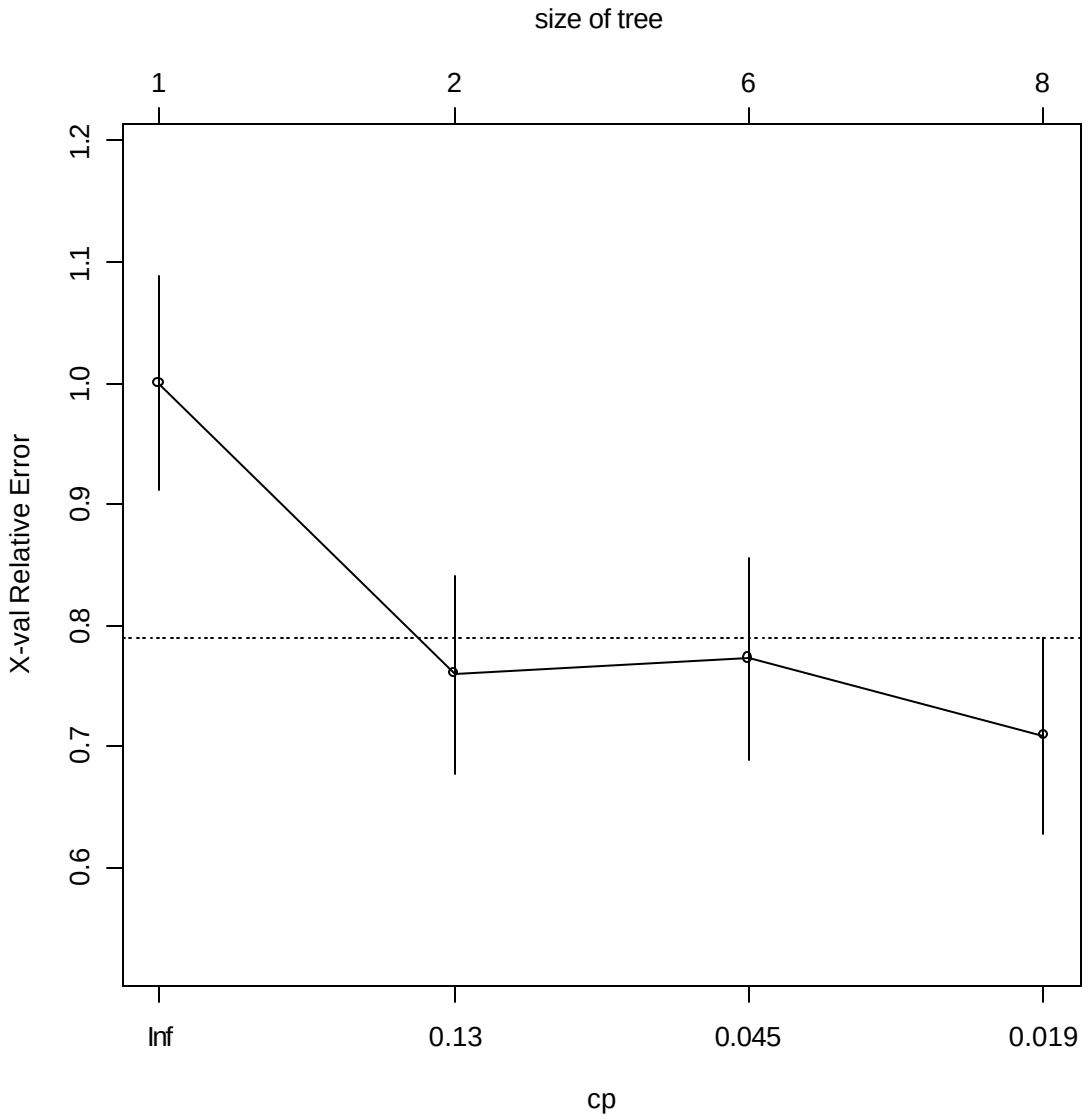
Classification tree:
rpart(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, data = data.training,
method = "class", parms = list(split = "information"))

Variables actually used in tree construction:
[1] V2 V3 V5 V6 V7 V8

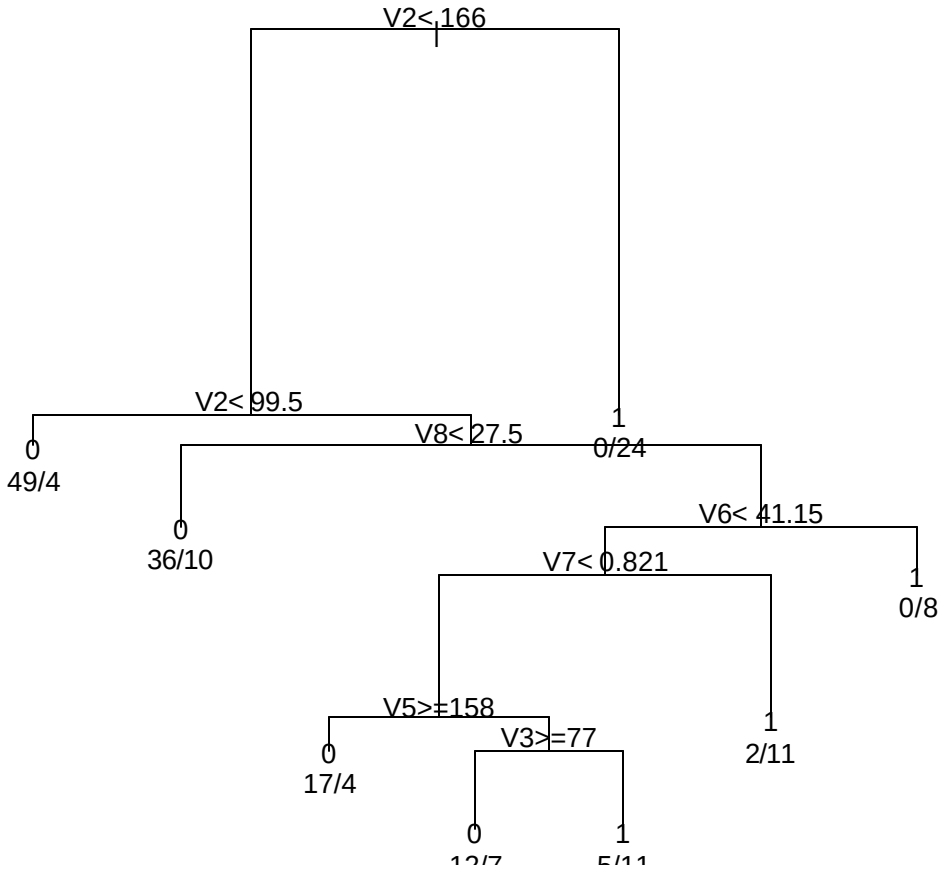
Root node error: 79/200 = 0.395

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.303797	0	1.000000	1.000000	0.08751
2	0.053797	1	0.696200	0.759490	0.08204
3	0.037975	5	0.481010	0.772150	0.08242
4	0.010000	7	0.405060	0.708860	0.08038



take cp=0.019 to minimize X-val Relative Error.




```
> tree.training.error.rate
[1] 0.15994
>
> tree.test.error.rate
[1] 0.26772
```

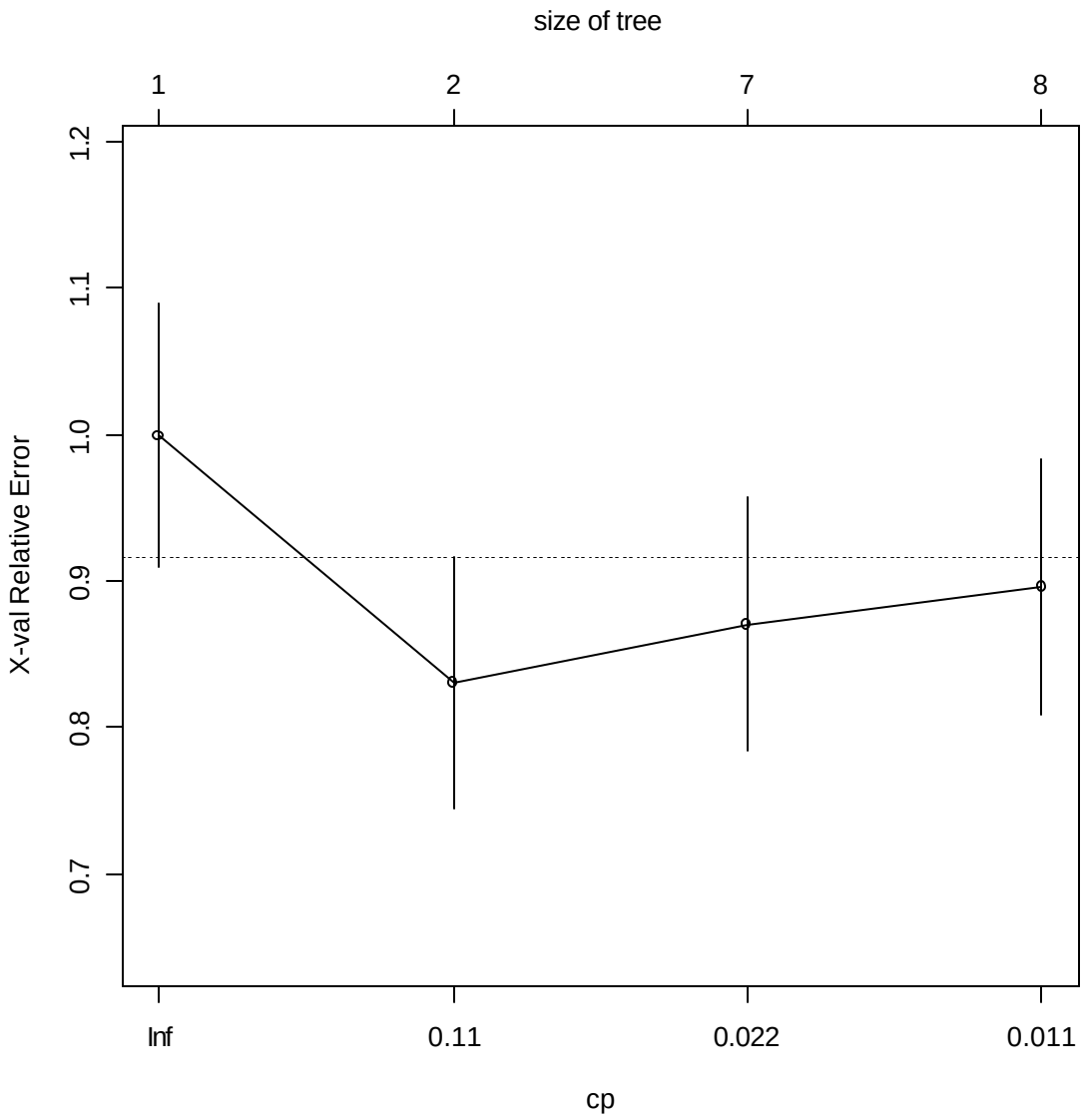
2.
Classification tree:
rpart(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, data = data.training,
    method = "class", parms = list(split = "information"))

Variables actually used in tree construction:
[1] V2 V3 V6 V8

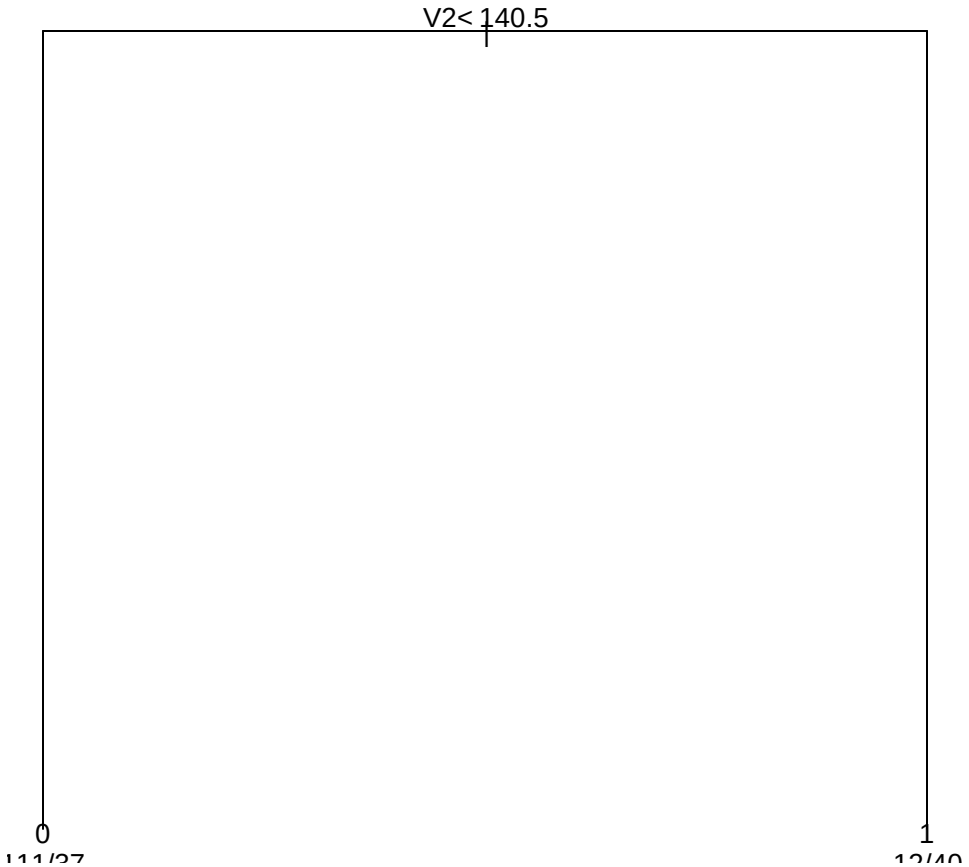
Root node error: 77/200 = 0.385

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.363636	0	1.000000	1.000000	0.08937
2	0.035714	1	0.636360	0.831170	0.08568
3	0.012987	6	0.441560	0.870130	0.08669
4	0.010000	7	0.428570	0.896100	0.08731



take cp=0.11 to minimize X-val Relative Error.




```
> tree.training.error.rate
[1] 0.24576
>
> tree.test.error.rate
[1] 0.25555
```

3.

Classification tree:

rpart(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, data = data.training,
method = "class", parms = list(split = "information"))

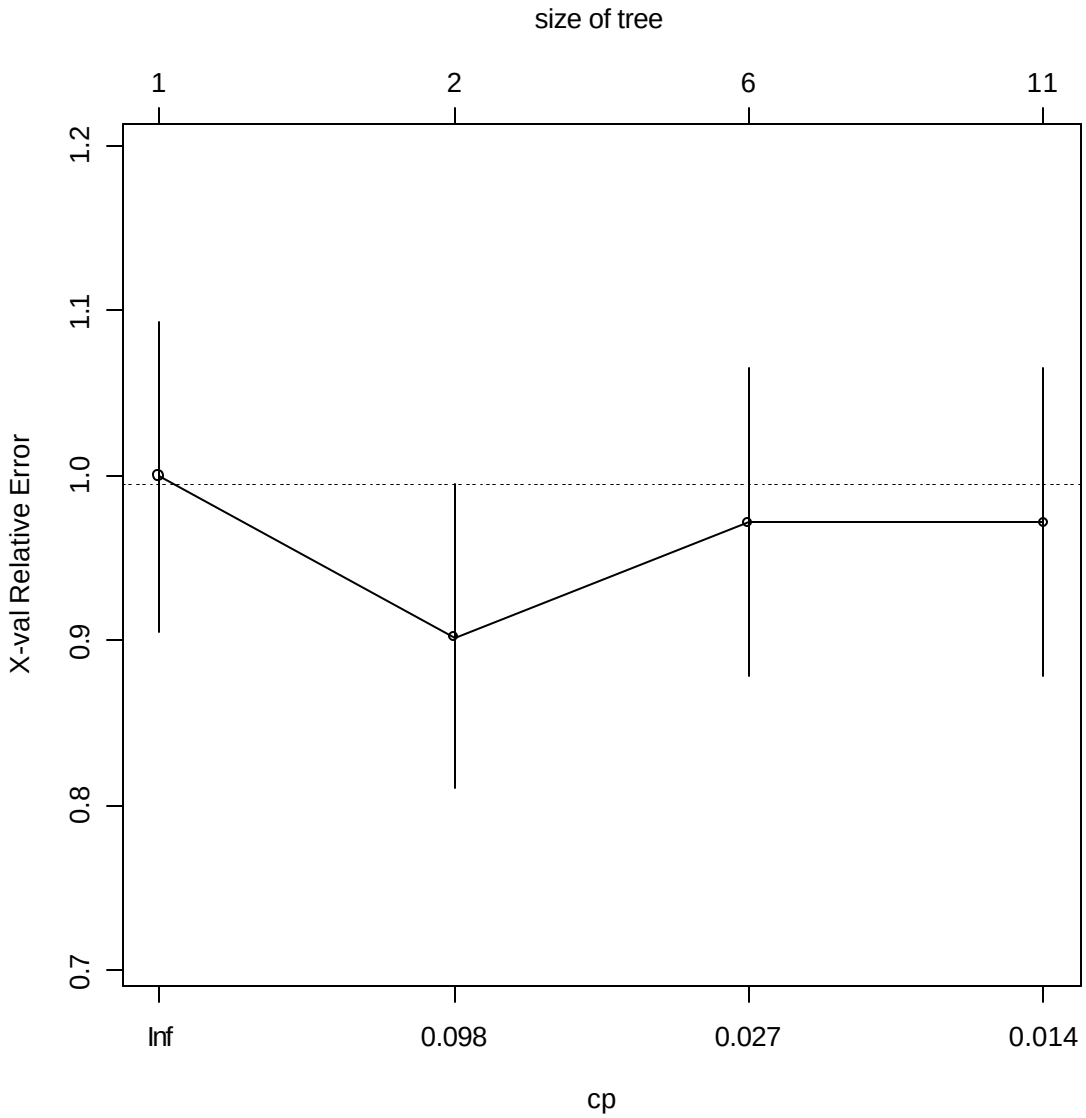
Variables actually used in tree construction:

[1] V2 V3 V6 V7 V8

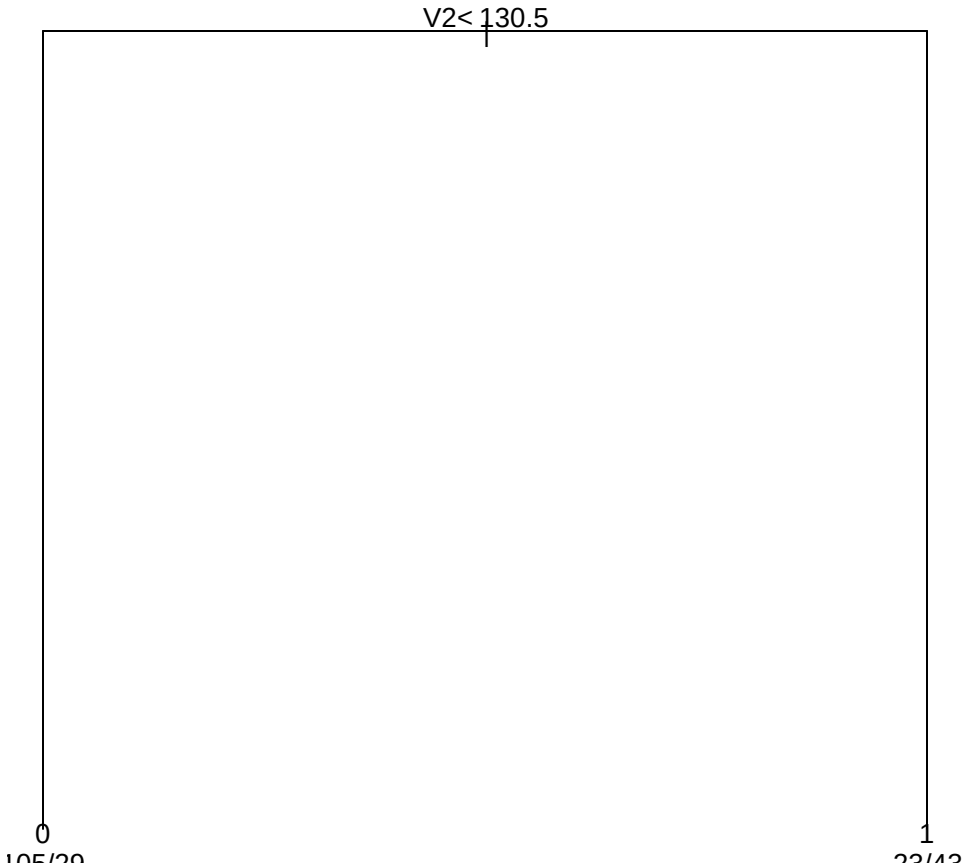
Root node error: 72/200 = 0.36

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.277778	0	1.000000	1.000000	0.09428
2	0.034722	1	0.722220	0.902780	0.09200
3	0.020833	5	0.569440	0.972220	0.09369
4	0.010000	10	0.458330	0.972220	0.09369



take cp=0.098 to minimize X-val Relative Error.




```
> tree.training.error.rate
[1] 0.25920
>
> tree.test.error.rate
[1] 0.26028
```

4.

Classification tree:

```
rpart(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, data = data.training,
    method = "class", parms = list(split = "information"))
```

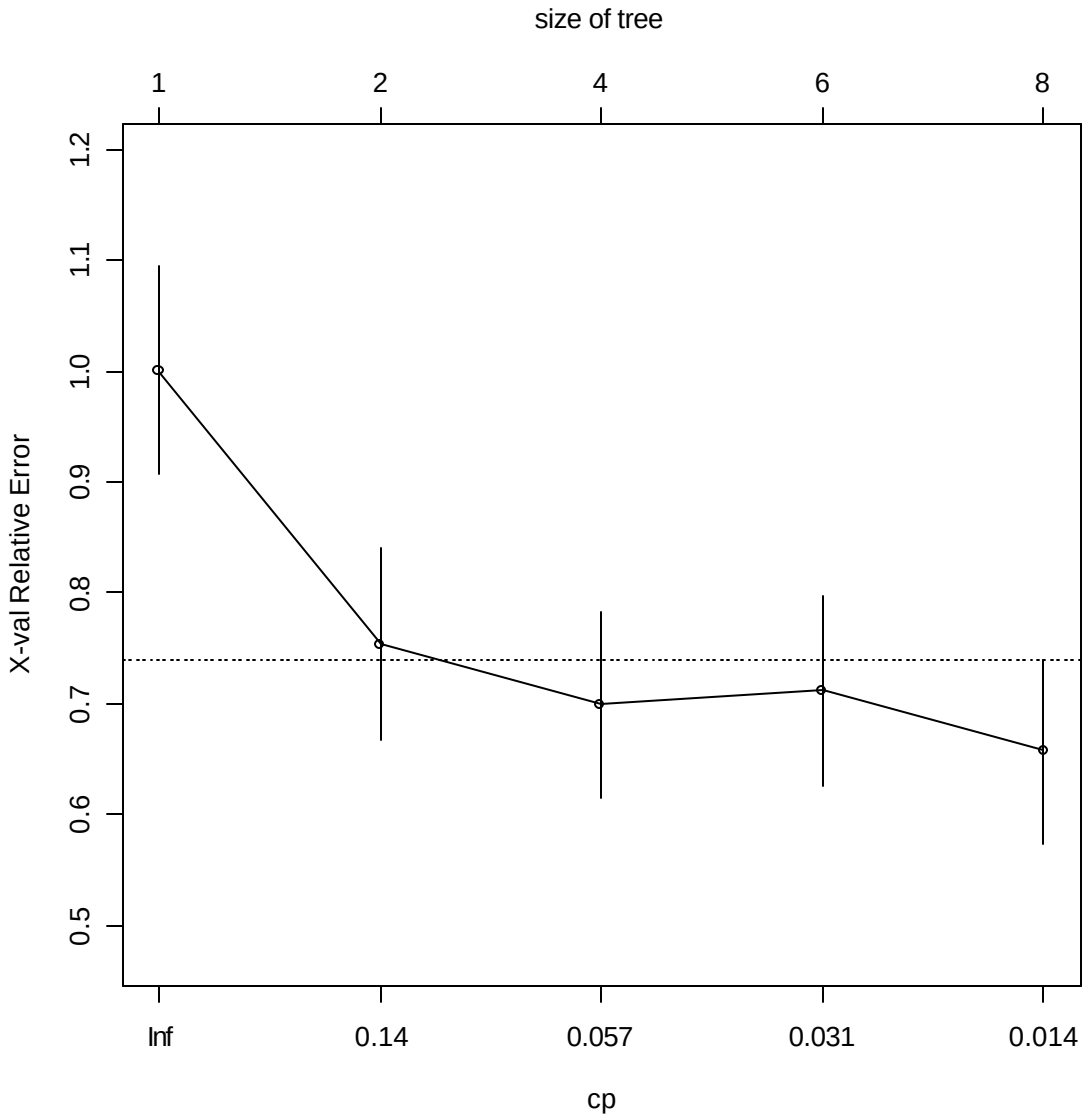
Variables actually used in tree construction:

[1] V2 V3 V6 V8

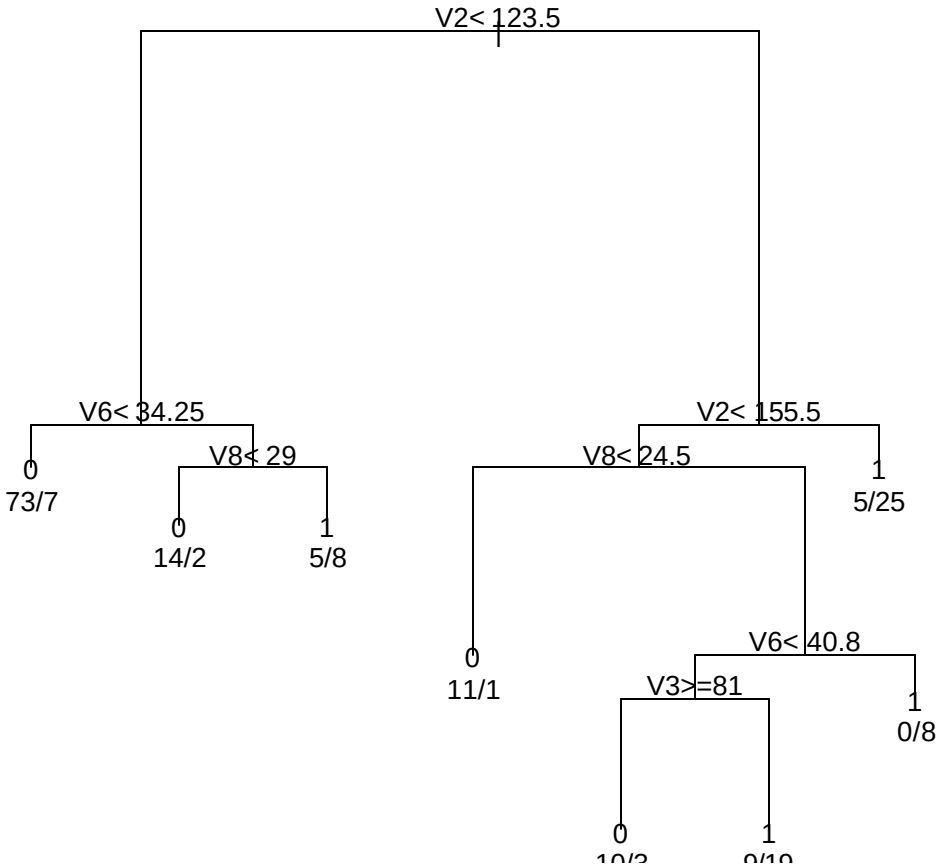
Root node error: 73/200 = 0.365

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.287671	0	1.000000	1.000000	0.09327
2	0.068493	1	0.712330	0.753420	0.08650
3	0.047945	3	0.575340	0.698630	0.08444
4	0.020548	5	0.479450	0.712330	0.08498
5	0.010000	7.000000	0.438360	0.657530	0.08274



take cp=0.014 to minimize X-val Relative Error.



> tree.training.error.rate
[1] 0.15936
>
> tree.test.error.rate
[1] 0.29070

5.

Classification tree:

```
rpart(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, data = data.training,
    method = "class", parms = list(split = "information"))
```

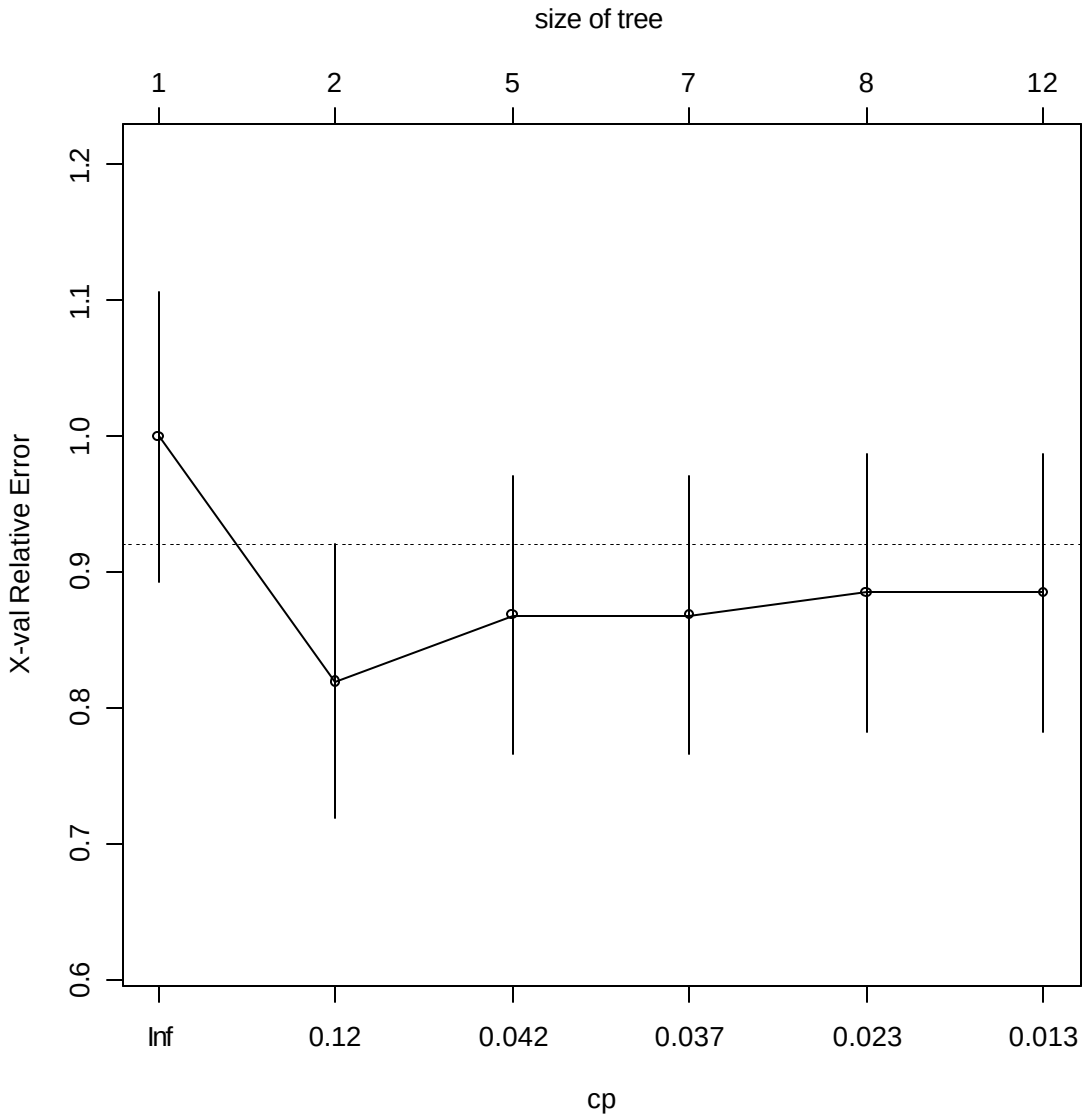
Variables actually used in tree construction:

[1] V2 V3 V6 V7 V8

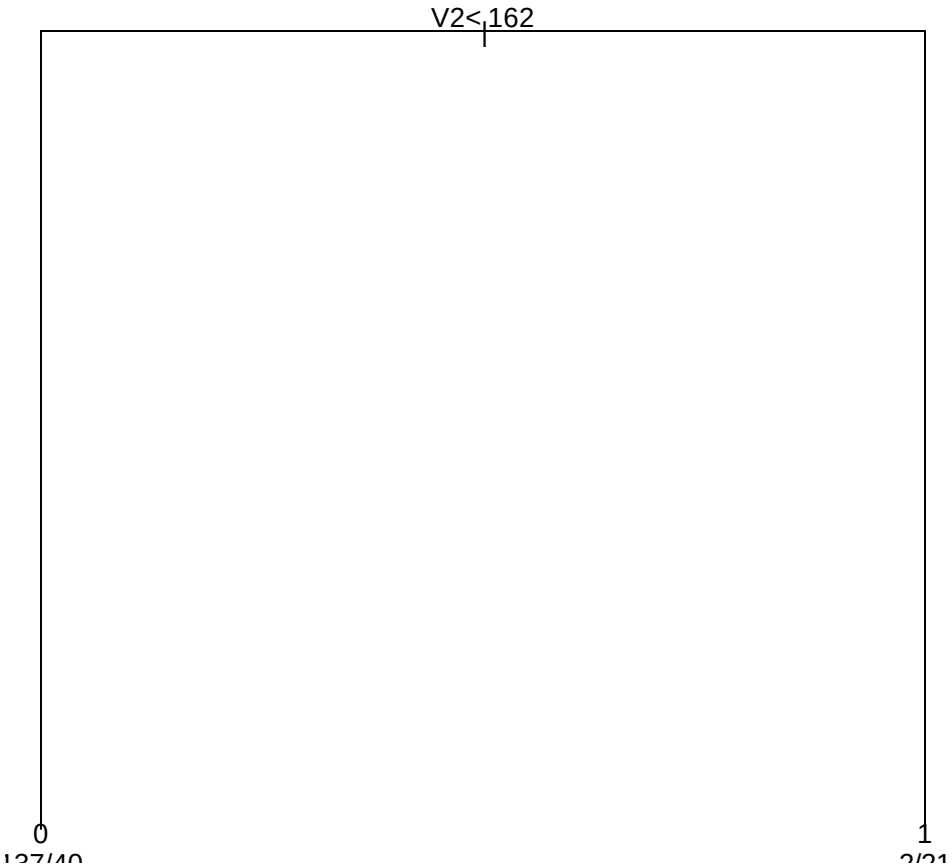
Root node error: 61/200 = 0.305

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.311475	0	1.000000	1.000000	0.10674
2	0.043716	1	0.688520	0.819670	0.10039
3	0.040984	4	0.557380	0.868850	0.10232
4	0.032787	6	0.475410	0.868850	0.10232
5	0.016393	7	0.442620	0.885250	0.10293
6	0.010000	11	0.377050	0.885250	0.10293



Take cp=0.12 to minimize X-val Relative Error.




```
> tree.training.error.rate
[1] 0.20928
>
> tree.test.error.rate
[1] 0.28327
```

6.

Classification tree:

```
rpart(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, data = data.training,
    method = "class", parms = list(split = "information"))
```

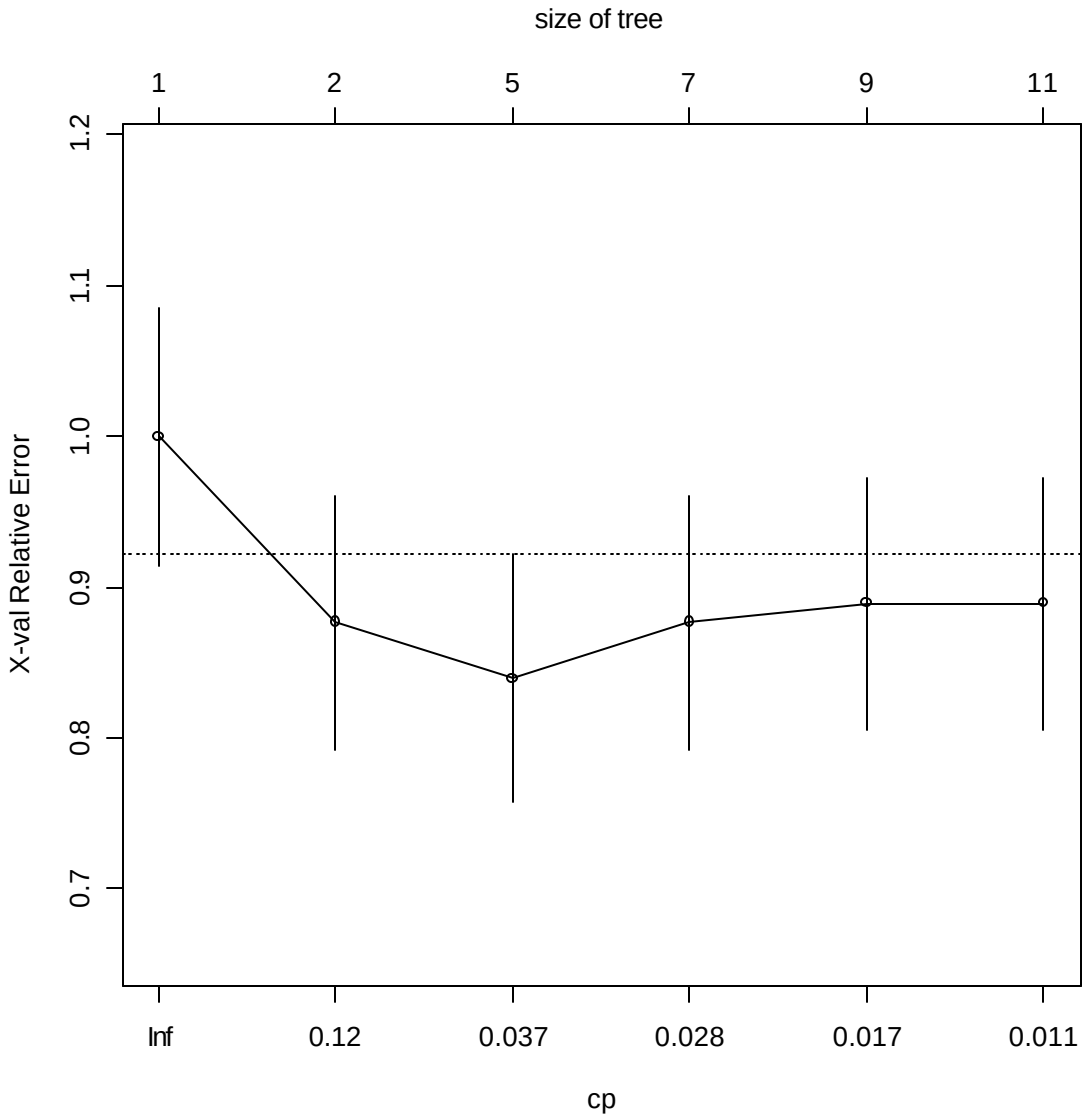
Variables actually used in tree construction:

[1] V1 V2 V6 V7 V8

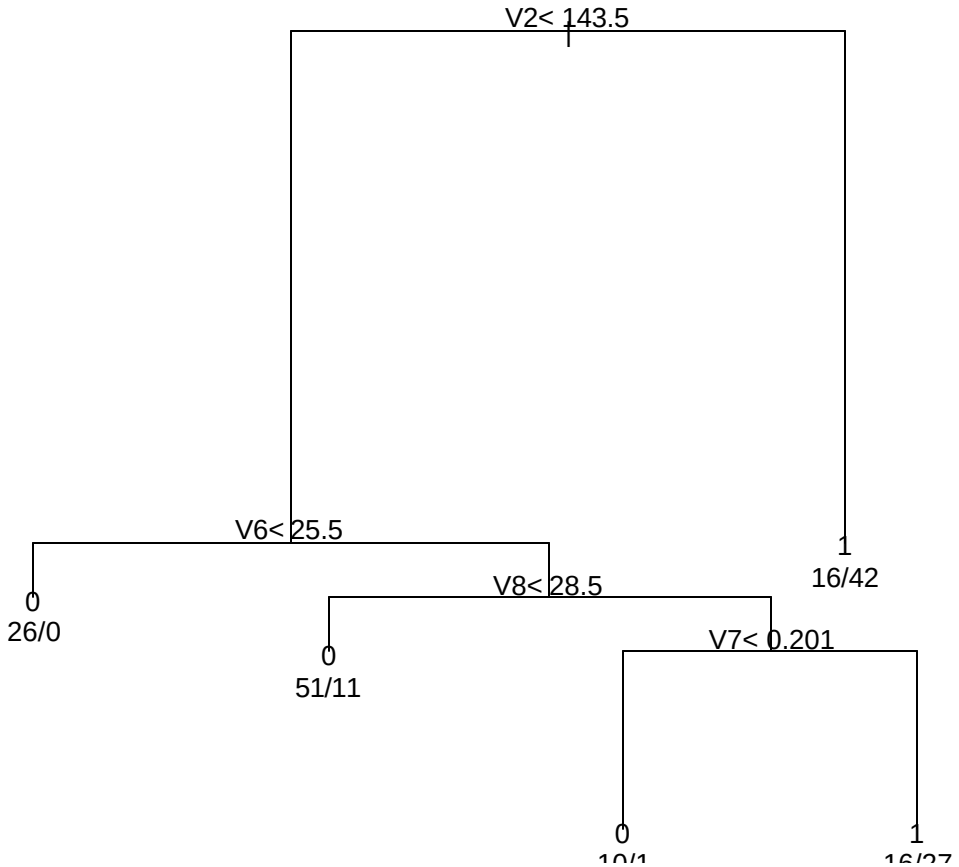
Root node error: 81/200 = 0.405

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.320988	0	1.000000	1.000000	0.08571
2	0.045267	1	0.679010	0.876540	0.08355
3	0.030864	4	0.543210	0.839510	0.08271
4	0.024691	6	0.481480	0.876540	0.08355
5	0.012346	8	0.432100	0.888890	0.08381
6	0.010000	10	0.407410	0.888890	0.08381



Take cp=0.037 to minimize X-val Relative Error.




```
> tree.training.error.rate
[1] 0.22080
>
> tree.test.error.rate
[1] 0.28868
```

7.

Classification tree:

```
rpart(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, data = data.training,
    method = "class", parms = list(split = "information"))
```

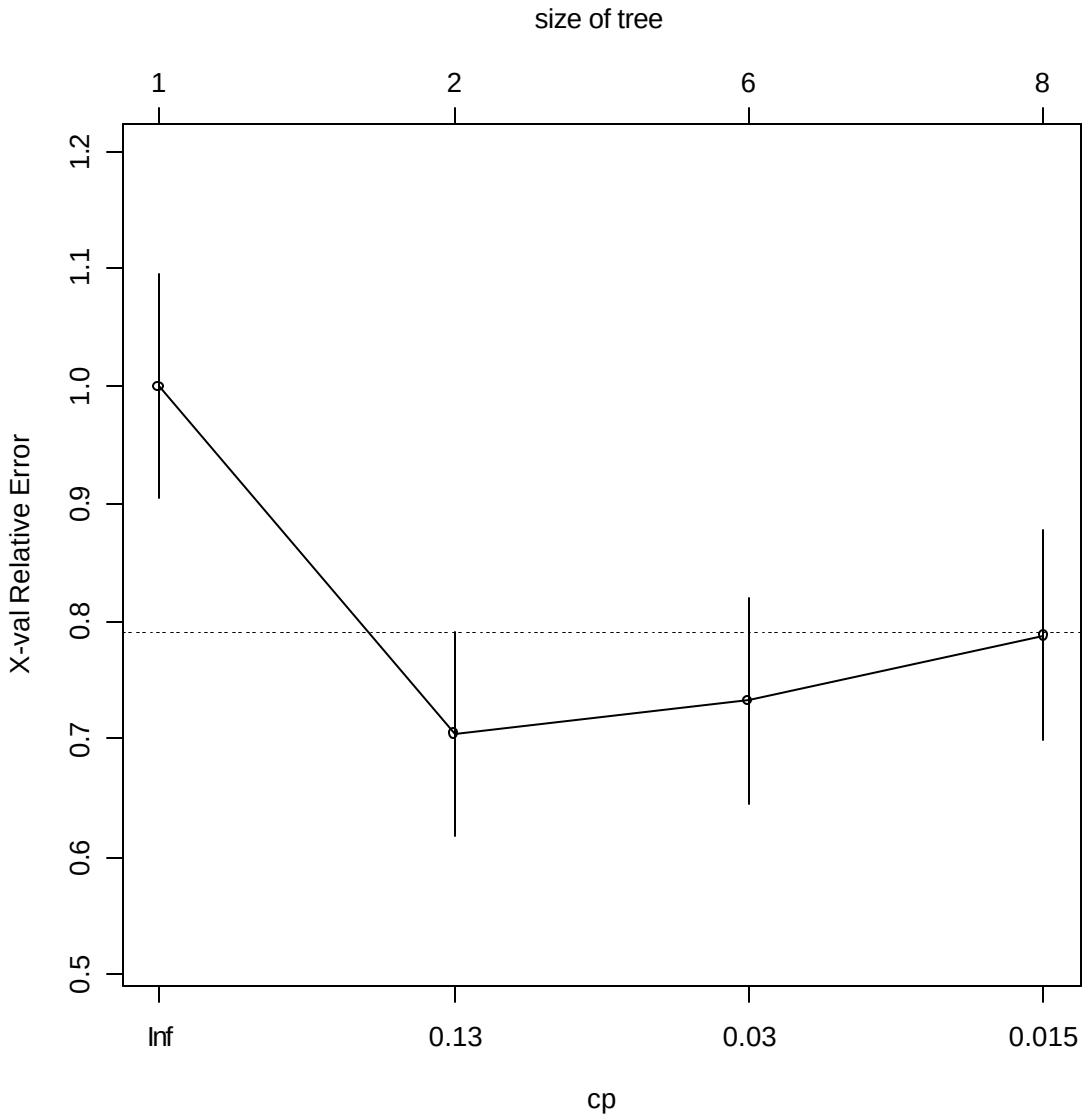
Variables actually used in tree construction:

[1] V2 V5 V6 V7 V8

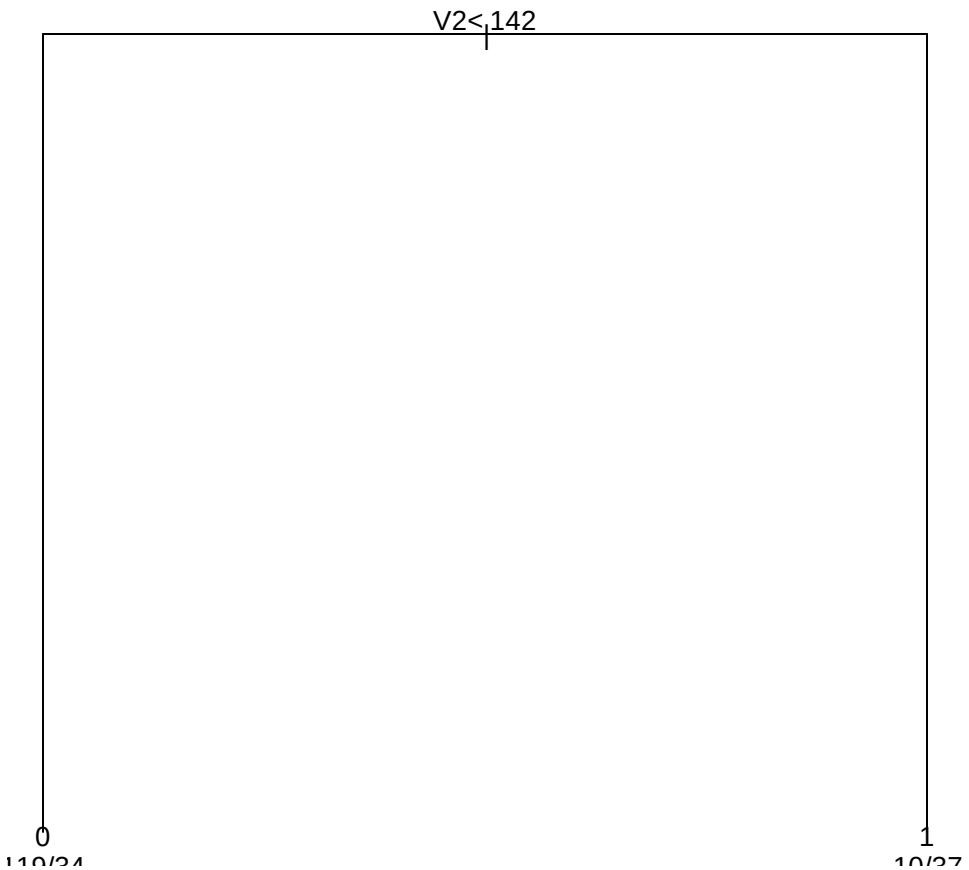
Root node error: 71/200 = 0.355

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.380282	0	1.000000	1.000000	0.09531
2	0.042254	1	0.619720	0.704230	0.08625
3	0.021127	5	0.436620	0.732390	0.08737
4	0.010000	7	0.394370	0.788730	0.08943



Take cp=0.13 to minimize X-val Relative Error.




```
> tree.training.error.rate
[1] 0.22080
>
> tree.test.error.rate
[1] 0.26231
```

8.

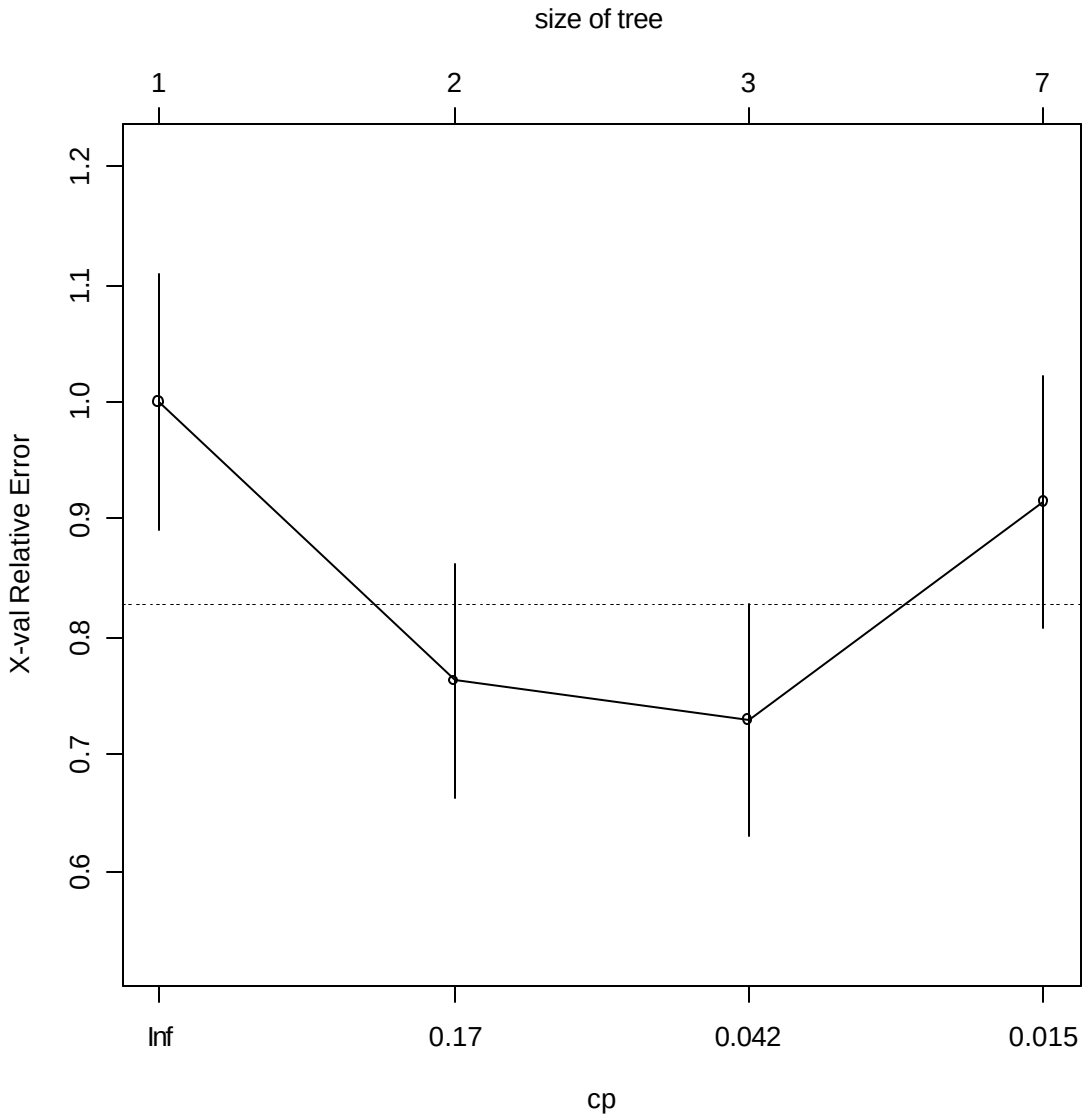
Classification tree:
rpart(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, data = data.training,
method = "class", parms = list(split = "information"))

Variables actually used in tree construction:
[1] V2 V5 V6 V8

Root node error: 59/200 = 0.295

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.338983	0	1.000000	1.000000	0.10931
2	0.084746	1	0.661020	0.762710	0.10009
3	0.021186	2	0.576270	0.728810	0.09847
4	0.010000	6	0.491530	0.915250	0.10642



Take cp=0.042 to minimize X-val Relative Error.


```
> tree.training.error.rate
[1] 0.17088
>
> tree.test.error.rate
[1] 0.27989
```

9.

Classification tree:

rpart(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, data = data.training,
method = "class", parms = list(split = "information"))

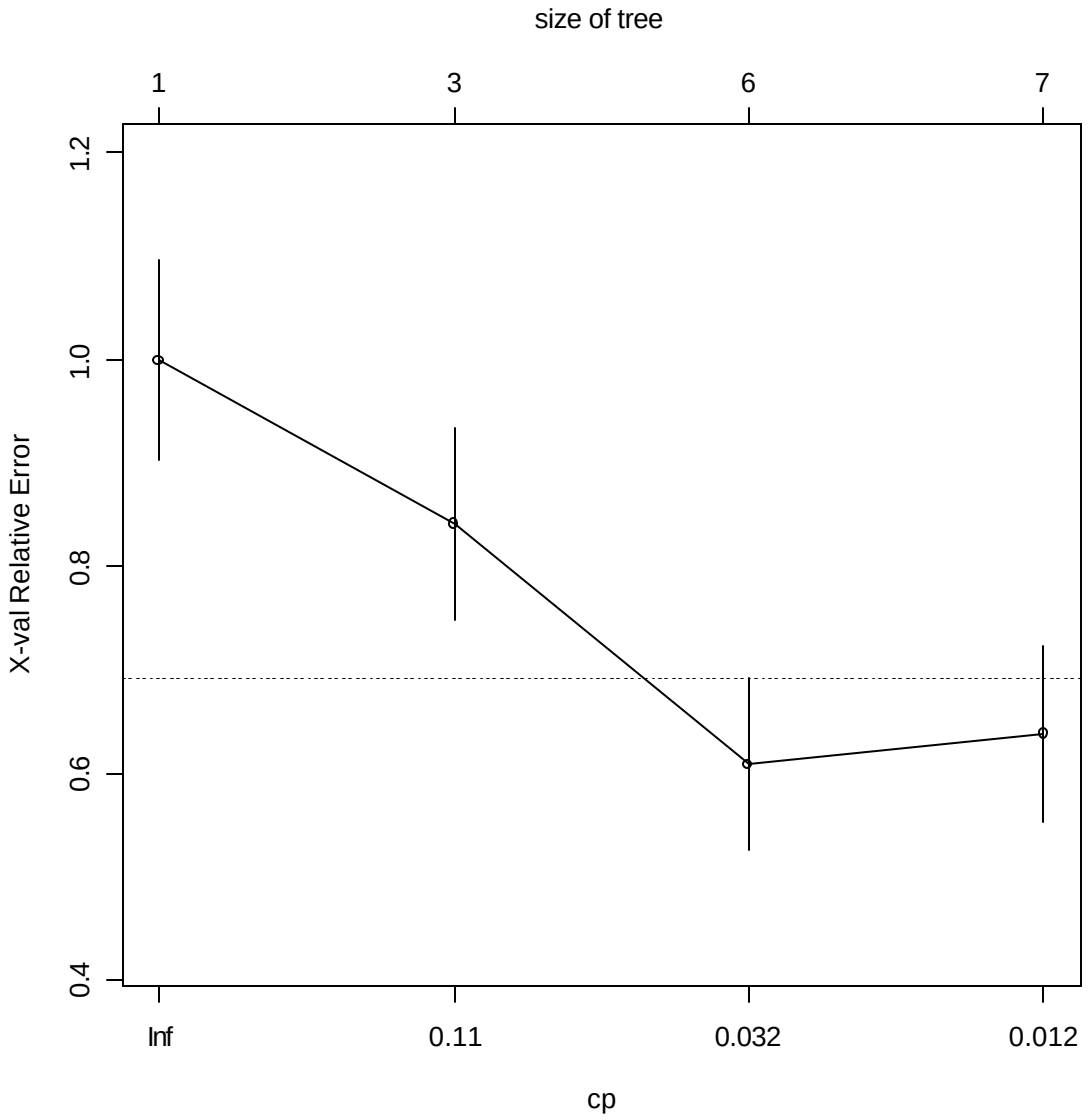
Variables actually used in tree construction:

[1] V2 V3 V6 V7 V8

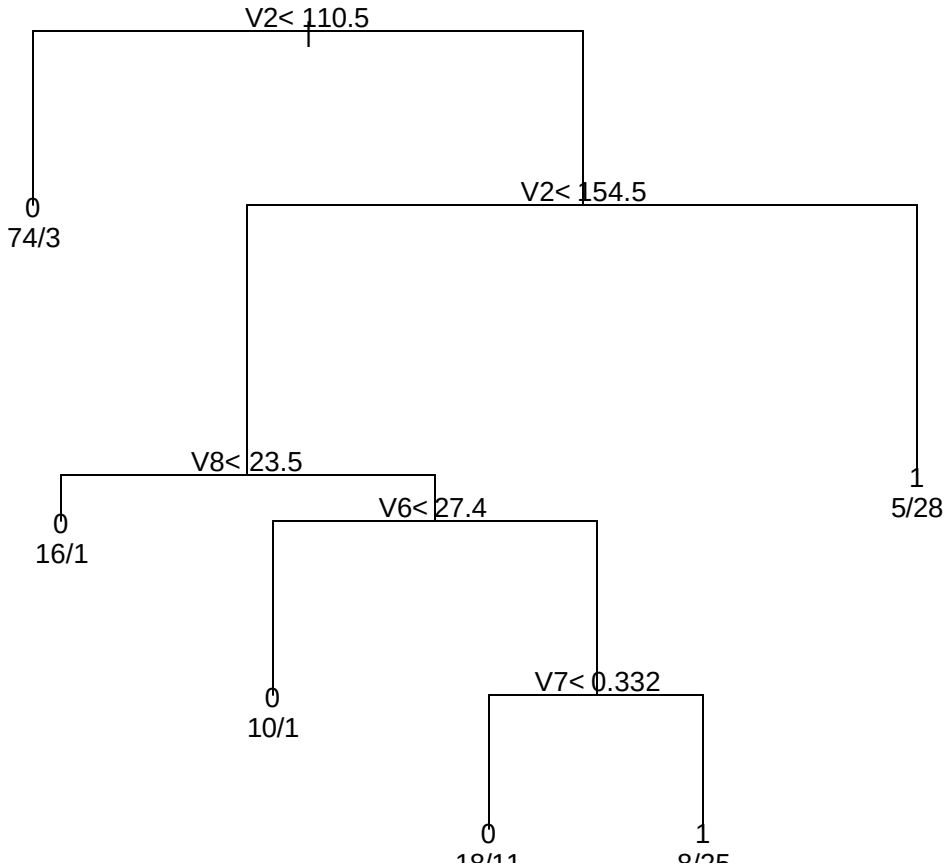
Root node error: 69/200 = 0.345

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.166667	0	1.000000	1.000000	0.09743
2	0.072464	2	0.666670	0.840580	0.09300
3	0.014493	5	0.420290	0.608700	0.08348
4	0.010000	6	0.405800	0.637680	0.08490



Take cp=0.032 to minimize X-val Relative Error.



> tree.training.error.rate
[1] 0.14496
>
> tree.test.error.rate
[1] 0.25690

10.

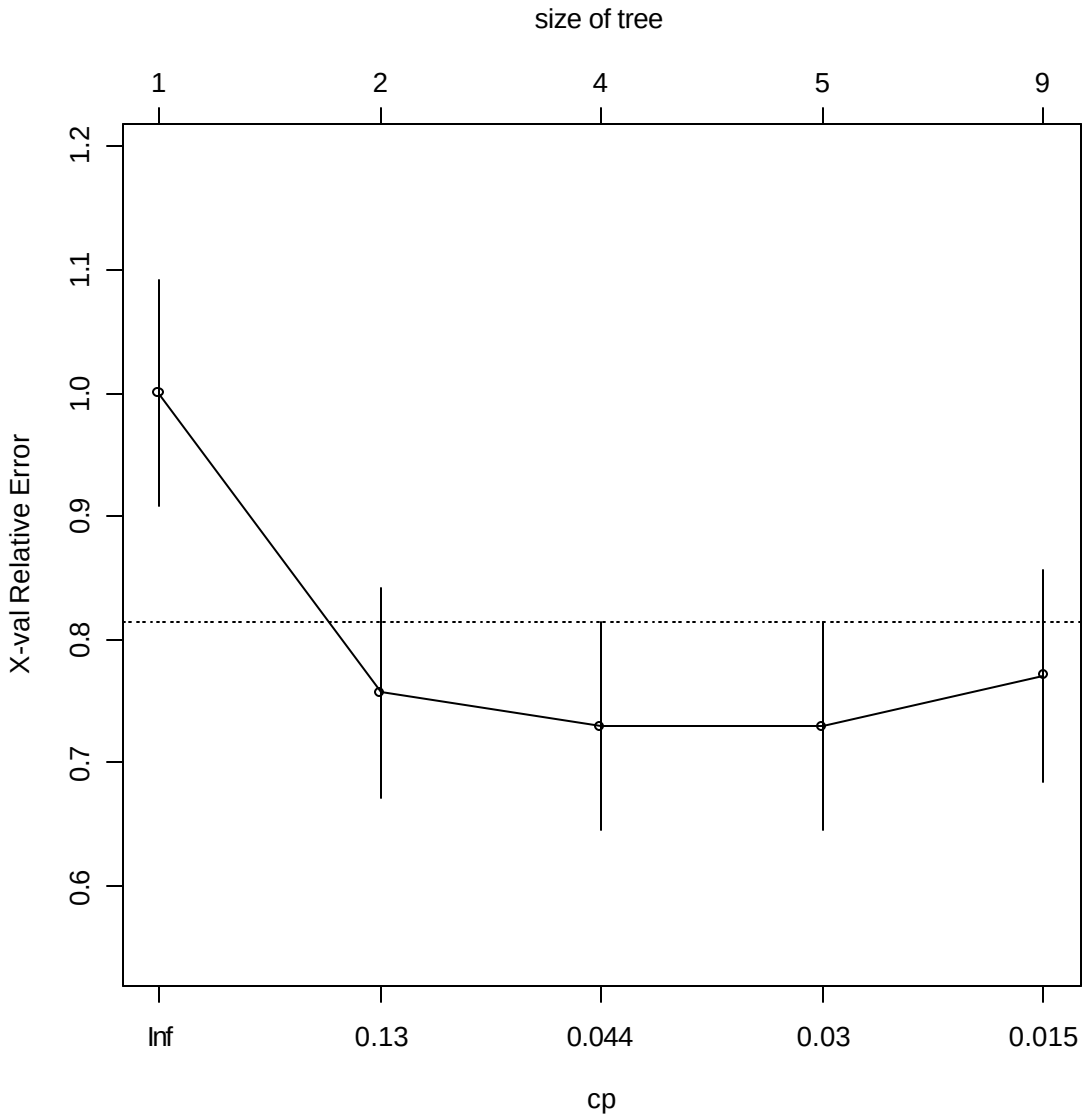
Classification tree:
rpart(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, data = data.training,
    method = "class", parms = list(split = "information"))

Variables actually used in tree construction:
[1] V2 V4 V5 V6 V7 V8

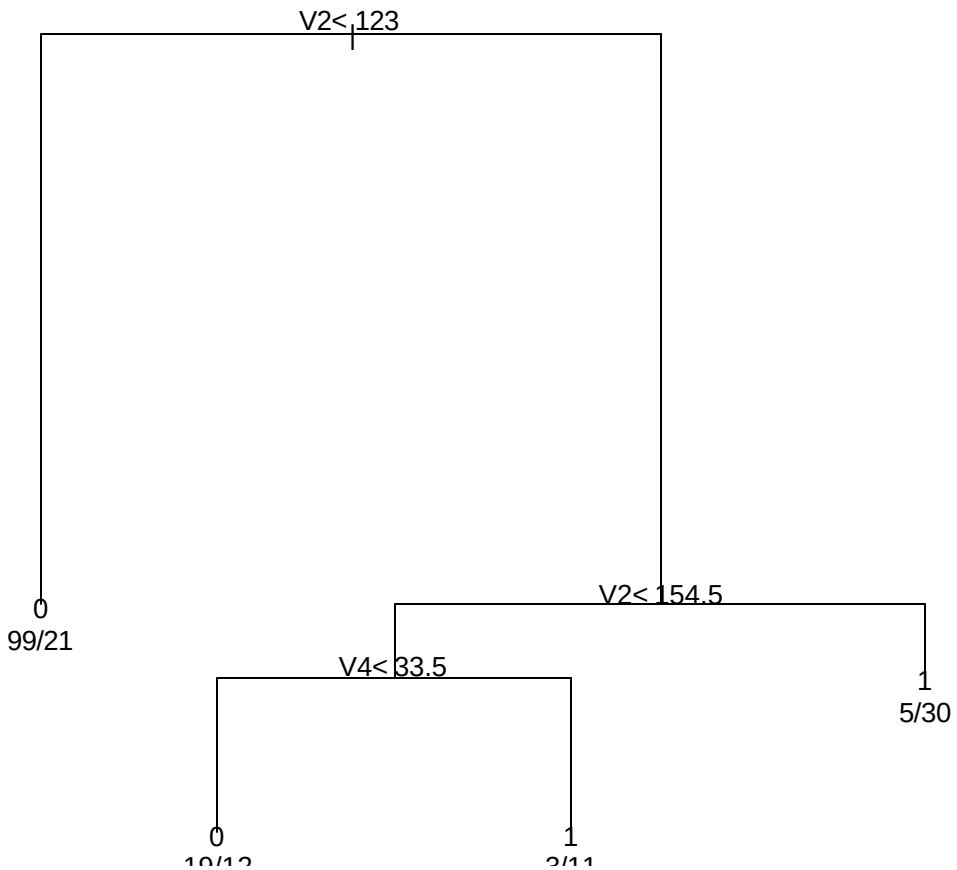
Root node error: 74/200 = 0.37

n= 200

	CP	nsplit	rel error	xerror	xstd
1	0.351351	0	1.000000	1.000000	0.09227
2	0.047297	1	0.648650	0.756760	0.08581
3	0.040541	3	0.554050	0.729730	0.08485
4	0.022523	4	0.513510	0.729730	0.08485
5	0.010000	8	0.405410	0.770270	0.08627



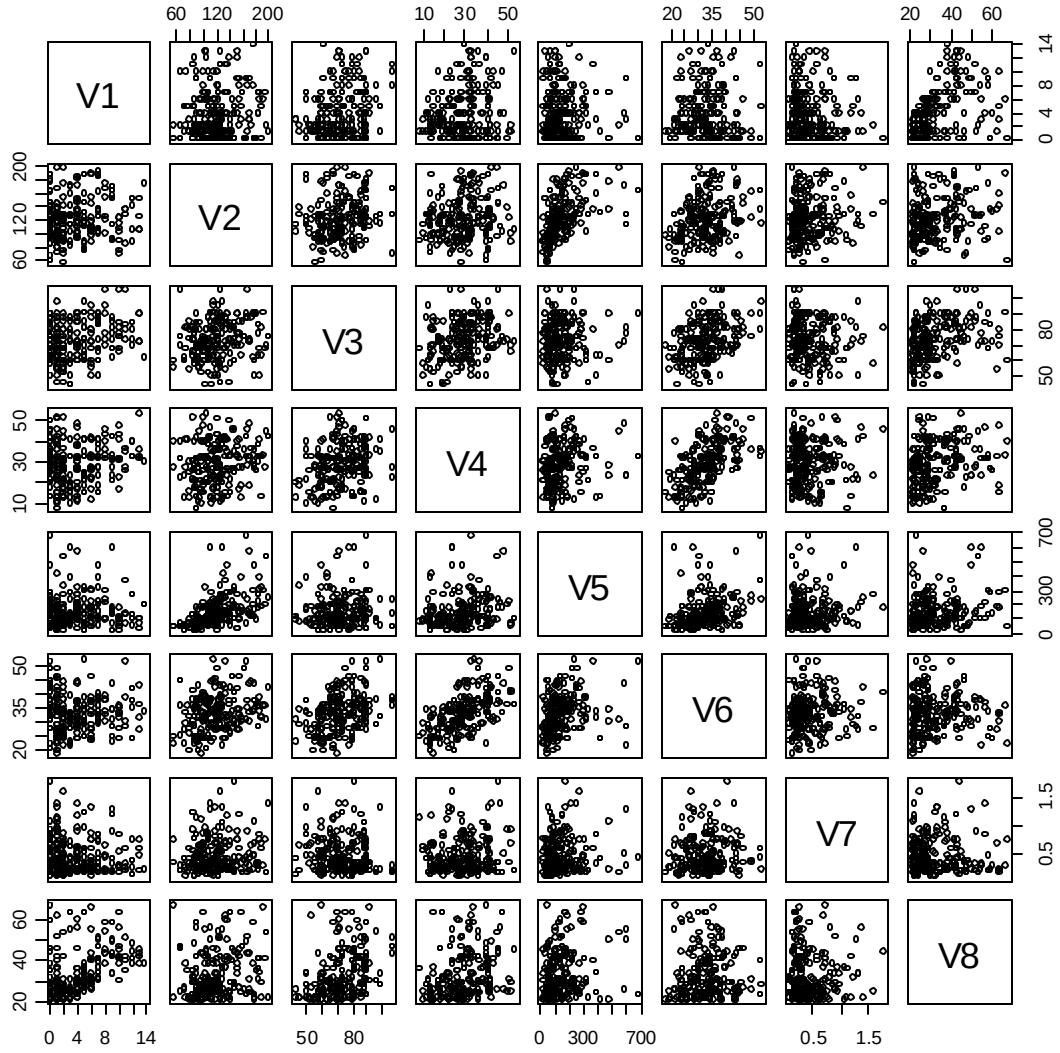
Take cp=0.044 to minimize X-val relative error.



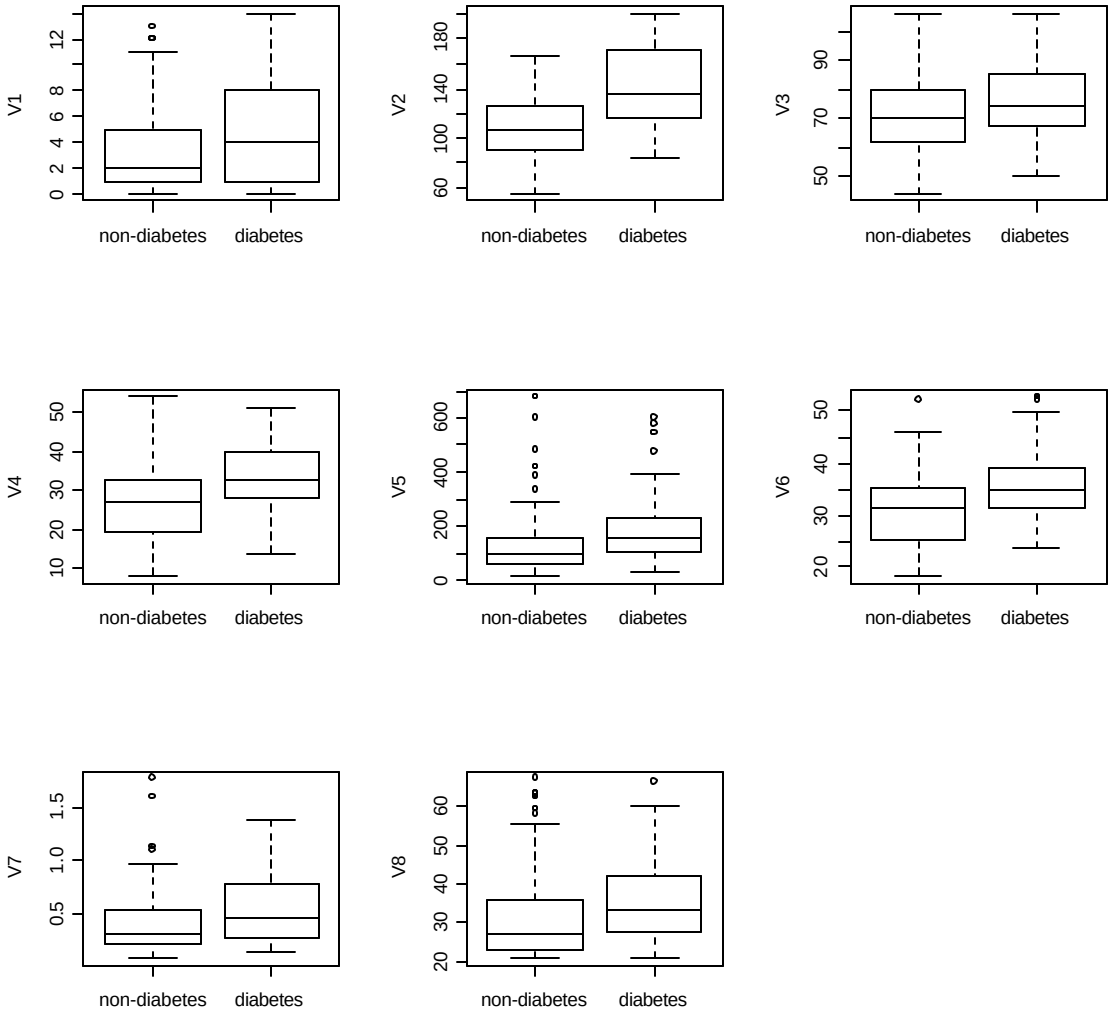

```
> tree.training.error.rate
[1] 0.20544
>
> tree.test.error.rate
[1] 0.24676
```

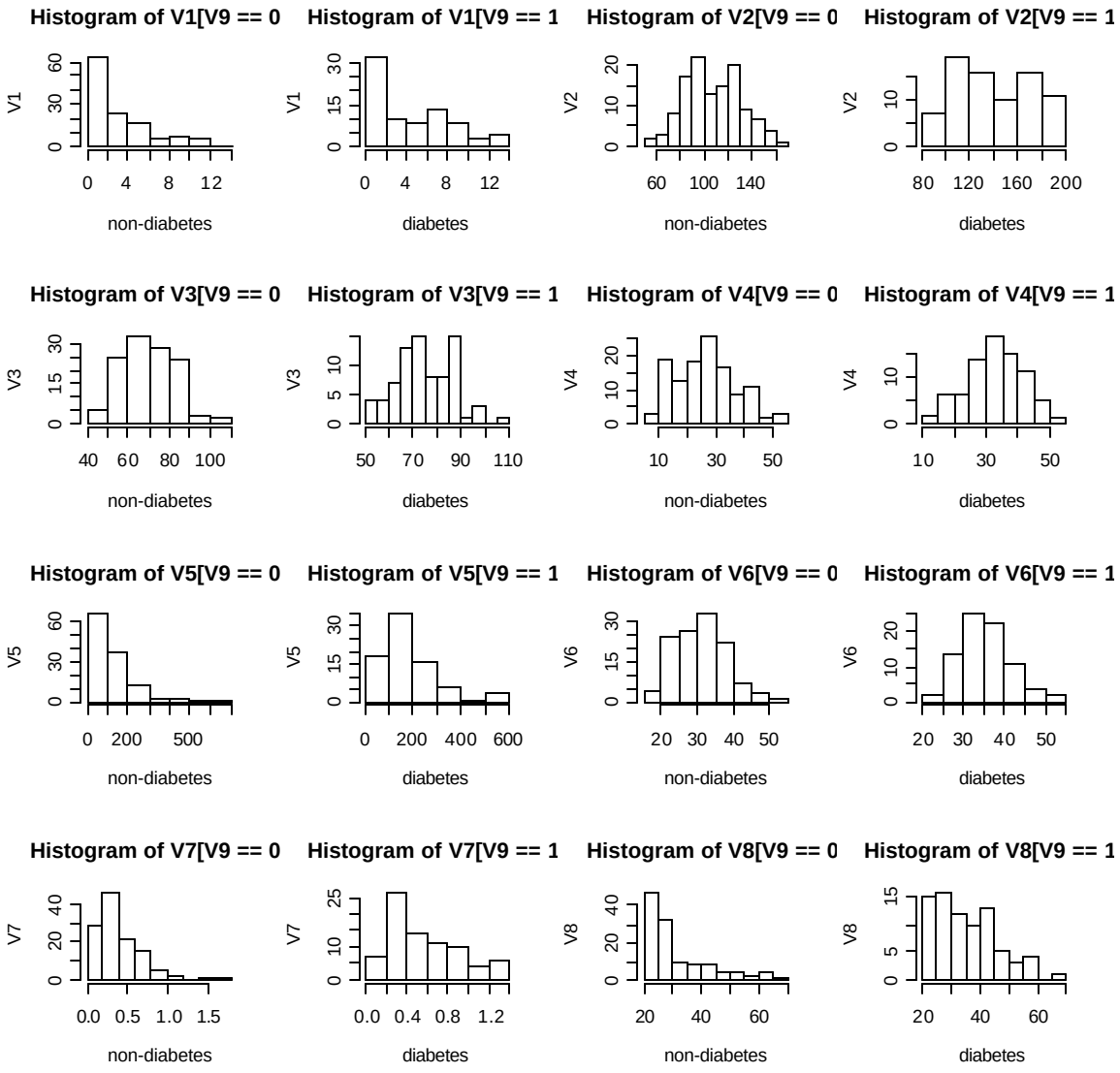
Method 1: Logistic Regression

A. Model Selection:



From the scatter plot, there seem to be some level of interaction between V4 and V6.


V1
0
2
4
6
8
12
non-diabetes
diabetes
V2
60
100
140
180
non-diabetes
diabetes
V3
50
70
90
non-diabetes
diabetes
V4
10
20
30
40
50
non-diabetes
diabetes
V5
0
200
400
600
non-diabetes
diabetes
V6
20
30
40
50
non-diabetes
diabetes
V7
0.5
1.0
1.5
non-diabetes
diabetes
V8
20
30
40
50
60
non-diabetes
diabetes


Histogram of V1[V9 == 0
V1
non-diabetes
Histogram of V1[V9 == 1
V1
diabetes
Histogram of V2[V9 == 0
V2
non-diabetes
Histogram of V2[V9 == 1
V2
diabetes
Histogram of V3[V9 == 0
V3
non-diabetes
Histogram of V3[V9 == 1
V3
diabetes
Histogram of V4[V9 == 0
V4
non-diabetes
Histogram of V4[V9 == 1
V4
diabetes
Histogram of V5[V9 == 0
V5
non-diabetes
Histogram of V5[V9 == 1
V5
diabetes
Histogram of V6[V9 == 0
V6
non-diabetes
Histogram of V6[V9 == 1
V6
diabetes
Histogram of V7[V9 == 0
V7
non-diabetes
Histogram of V7[V9 == 1
V7
diabetes
Histogram of V8[V9 == 0
V8
non-diabetes
Histogram of V8[V9 == 1
V8
diabetes

Step 1:

Fit the full model with possible interaction terms:

```
Call:
glm(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V4 *
    V6, family = binomial)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-2.7183  -0.6792  -0.3148   0.6244   2.2186

Coefficients:
             Estimate  Std. Error  z value     Pr(>|z|)
(Intercept) -10.88000     3.88800  -2.79800  0.00514 **
V1            0.09718     0.06266   1.55100  0.12095
V2            0.04546     0.00845   5.38200  0.00000 ***
V3           -0.01694     0.01772  -0.95600  0.33918
V4            0.04373     0.11900   0.36800  0.71324
V5           -0.00264     0.00196  -1.34600  0.17815
V6            0.11340     0.11890   0.95300  0.34040
V7            1.91700     0.61560   3.11500  0.00184 **
V8            0.01987     0.02106   0.94400  0.34542
V4:V6        -0.00060     0.00347  -0.17300  0.86279
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 268.37  on 199  degrees of freedom
Residual deviance: 178.10  on 190  degrees of freedom
AIC: 198.1

Number of Fisher Scoring iterations: 5
```

Step 2:

Since the interaction is not significant, drop it.

```
Call:
glm(formula = V9 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8, family = binomial)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
```

-2.7300 -0.6751 -0.3224 0.6273 2.2355

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-10.280954	1.727404	-5.952000	0.000000	***
V1	0.098330	0.062403	1.576000	0.115090	
V2	0.045646	0.008380	5.447000	0.000000	***
V3	-0.016883	0.017718	-0.953000	0.340650	
V4	0.023561	0.022636	1.041000	0.297960	
V5	-0.002670	0.001948	-1.371000	0.170410	
V6	0.094103	0.040445	2.327000	0.019980	*
V7	1.911595	0.614188	3.112000	0.001860	**
V8	0.020094	0.021031	0.955000	0.339360	

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 268.37 on 199 degrees of freedom
Residual deviance: 178.13 on 191 degrees of freedom
AIC: 196.13

Number of Fisher Scoring iterations: 5

Step 3:

Drop all the features with p-value>0.12

Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:
Min 1Q Median 3Q Max
-2.5043 -0.7232 -0.3307 0.6982 2.1453

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-10.017692	1.500661	-6.676000	0.000000	***
V1	0.126211	0.052323	2.412000	0.015860	*
V2	0.040658	0.007344	5.536000	0.000000	***
V6	0.093280	0.030352	3.073000	0.002120	**
V7	1.906364	0.594621	3.206000	0.001350	**

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 268.37 on 199 degrees of freedom
Residual deviance: 182.90 on 195 degrees of freedom
AIC: 192.90

Number of Fisher Scoring iterations: 5

Step 4:

Add feature V3:

Call:
glm(formula = V9 ~ V1 + V2 + V3 + V6 + V7, family = binomial)

Deviance Residuals:
    Min   1Q   Median   3Q   Max
-2.4695 -0.7204 -0.3366   0.7305   2.2100

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-9.629108	1.610551	-5.979000	0.000000	***
V1	0.138404	0.055919	2.475000	0.013320	*
V2	0.041386	0.007507	5.513000	0.000000	***
V3	-0.010781	0.016453	-0.655000	0.512310	
V6	0.101514	0.033100	3.067000	0.002160	**
V7	1.886691	0.595938	3.166000	0.001550	**

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 268.37 on 199 degrees of freedom
Residual deviance: 182.47 on 194 degrees of freedom
AIC: 194.47

Number of Fisher Scoring iterations: 5

Step 5:

Drop V3 and add V4:
Call:
glm(formula = V9 ~ V1 + V2 + V4 + V6 + V7, family = binomial)

Deviance Residuals:
    Min   1Q   Median   3Q   Max

-2.6094 -0.7187 -0.3209 0.6378 2.2229

Coefficients:

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-9.984900	1.493926	-6.684000	0.000000	***
V1	0.109028	0.053252	2.047000	0.040620	*
V2	0.041099	0.007427	5.533000	0.000000	***
V4	0.030348	0.022145	1.370000	0.170570	
V6	0.066486	0.035502	1.873000	0.061110	.
V7	1.834050	0.599769	3.058000	0.002230	**

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 268.37 on 199 degrees of freedom
Residual deviance: 180.99 on 194 degrees of freedom
AIC: 192.99

Number of Fisher Scoring iterations: 5

Step 6:

Drop V4 and add V5:

Call:
glm(formula = V9 ~ V1 + V2 + V5 + V6 + V7, family = binomial)

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.5726	-0.7131	-0.3377	0.6223	2.0707

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-10.434045	1.555883	-6.706000	0.000000	***
V1	0.125541	0.052829	2.376000	0.017485	*
V2	0.045074	0.008248	5.465000	0.000000	***
V5	-0.002422	0.001892	-1.280000	0.200527	
V6	0.099784	0.030939	3.225000	0.001259	**
V7	1.976300	0.600497	3.291000	0.000998	***

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 268.37 on 199 degrees of freedom
Residual deviance: 181.23 on 194 degrees of freedom
AIC: 193.23

Number of Fisher Scoring iterations: 5

Step 7:

Drop V5 and add V8:
Call:
glm(formula = V9 ~ V1 + V2 + V8 + V6 + V7, family = binomial)

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.5974	-0.7230	-0.3189	0.6662	2.1568

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-10.353637	1.580217	-6.552000	0.000000	***
V1	0.103782	0.059757	1.737000	0.082430	.
V2	0.039854	0.007389	5.394000	0.000000	***
V8	0.014400	0.019157	0.752000	0.452250	
V6	0.094228	0.030298	3.110000	0.001870	**
V7	1.922217	0.598967	3.209000	0.001330	**

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 268.37 on 199 degrees of freedom
Residual deviance: 182.34 on 194 degrees of freedom
AIC: 194.34

Number of Fisher Scoring iterations: 5

Conclusion:

Based on AIC value and the significance of each input features, we choose the following model, the one we tried in step 3:
Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Run the logistic regression classifier for 10 different training set and testing set for training error and testing error:

1 Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.5043	-0.7232	-0.3307	0.6982	2.1453

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-10.017692	1.500661	-6.676000	0.000000	***
V1	0.126211	0.052323	2.412000	0.015860	*
V2	0.040658	0.007344	5.536000	0.000000	***
V6	0.093280	0.030352	3.073000	0.002120	**
V7	1.906364	0.594621	3.206000	0.001350	**

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 268.37 on 199 degrees of freedom
Residual deviance: 182.90 on 195 degrees of freedom
AIC: 192.90

Number of Fisher Scoring iterations: 5

> training.error.rate
[1] 0.20928
> test.error.rate
[1] 0.23392

2 Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.3537	-0.7576	-0.4308	0.6957	2.3295

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-7.971804	1.237139	-6.444000	0.000000	***

V1	0.133110	0.050269	2.648000	0.008100 **
V2	0.039040	0.006883	5.672000	0.000000 ***
V6	0.055946	0.027385	2.043000	0.041100 *
V7	0.598001	0.638106	0.937000	0.348700

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 266.58 on 199 degrees of freedom
Residual deviance: 194.46 on 195 degrees of freedom
AIC: 204.46

Number of Fisher Scoring iterations: 4

> training.error.rate
[1] 0.24576
> test.error.rate
[1] 0.22377

3 Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:
Min 1Q Median 3Q Max
-2.1100 -0.8001 -0.4760 0.7970 2.1248

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-7.362406	1.203947	-6.115000	0.000000 ***
V1	0.126321	0.050820	2.486000	0.012900 *
V2	0.041232	0.007233	5.701000	0.000000 ***
V6	0.032287	0.027516	1.173000	0.240600
V7	0.181212	0.529618	0.342000	0.732200

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 261.37 on 199 degrees of freedom
Residual deviance: 202.35 on 195 degrees of freedom
AIC: 212.35

Number of Fisher Scoring iterations: 4

> training.error.rate

[1] 0.24960
> test.error.rate
[1] 0.23392

4 Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:
Min	1Q	Median	3Q	Max
-2.4471	-0.7668	-0.4138	0.8073	2.1332

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-8.751852	1.318556	-6.637000	0.000000	***
V1	0.112153	0.049340	2.273000	0.023020	*
V2	0.036095	0.006829	5.286000	0.000000	***
V6	0.084921	0.027302	3.110000	0.001870	**
V7	0.651337	0.564445	1.154000	0.248520	

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 262.50 on 199 degrees of freedom
Residual deviance: 195.01 on 195 degrees of freedom
AIC: 205.01

Number of Fisher Scoring iterations: 4

> training.error.rate
[1] 0.24000
> test.error.rate
[1] 0.22175

5 Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:
Min	1Q	Median	3Q	Max
-2.5614	-0.6527	-0.3988	0.5818	2.2898

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-9.306638	1.375238	-6.767000	0.000000	***
V1	0.202857	0.056832	3.569000	0.000358	***
V2	0.035260	0.006943	5.078000	0.000000	***

V6	0.085924	0.029760	2.887000	0.003886 **
V7	0.875813	0.574854	1.524000	0.127623

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 246.02 on 199 degrees of freedom
Residual deviance: 174.94 on 195 degrees of freedom
AIC: 184.94

Number of Fisher Scoring iterations: 5

> training.error.rate
[1] 0.21504
> test.error.rate
[1] 0.36439

6 Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:
Min 1Q Median 3Q Max
-2.1003 -0.8415 -0.5603 0.9679 2.0994

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-6.279039	1.099895	-5.709000	0.000000 ***
V1	0.046482	0.048663	0.955000	0.339500
V2	0.029134	0.005818	5.007000	0.000001 ***
V6	0.056115	0.025071	2.238000	0.025200 *
V7	0.164473	0.450306	0.365000	0.714900

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 269.99 on 199 degrees of freedom
Residual deviance: 224.97 on 195 degrees of freedom
AIC: 234.97

Number of Fisher Scoring iterations: 4

> training.error.rate
[1] 0.28032

```
> test.error.rate
[1] 0.20755
```

7 Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.1942	-0.7035	-0.3751	0.6001	2.4117

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-9.411391	1.448659	-6.497000	0.000000	***
V1	0.207559	0.057270	3.624000	0.000290	***
V2	0.035149	0.006736	5.218000	0.000000	***
V6	0.094416	0.031844	2.965000	0.003030	**
V7	1.081381	0.549653	1.967000	0.049140	*

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 260.19 on 199 degrees of freedom
Residual deviance: 177.78 on 195 degrees of freedom
AIC: 187.78

Number of Fisher Scoring iterations: 5

```
> training.error.rate
[1] 0.20736
> test.error.rate
[1] 0.23932
```

8 Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.1544	-0.6770	-0.3632	0.5102	2.6327

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-10.646250	1.570600	-6.778000	0.000000	***
V1	0.149020	0.066210	2.251000	0.024410	*
V2	0.043180	0.007440	5.804000	0.000000	***
V6	0.102810	0.029030	3.541000	0.000398	***

V7 0.846850 0.644400 1.314000 0.188789

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 242.63 on 199 degrees of freedom
Residual deviance: 166.80 on 195 degrees of freedom
AIC: 176.80

Number of Fisher Scoring iterations: 5

> training.error.rate
[1] 0.18048
> test.error.rate
[1] 0.24811

9 Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:
Min 1Q Median 3Q Max
-2.4873 -0.6906 -0.3736 0.6929 2.4915

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-8.800978	1.296943	-6.786000	0.000000	***
V1	0.096536	0.053309	1.811000	0.070200	.
V2	0.040088	0.007184	5.581000	0.000000	***
V6	0.068916	0.027218	2.532000	0.011300	*
V7	0.767113	0.537204	1.428000	0.153300	

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 257.72 on 199 degrees of freedom
Residual deviance: 181.73 on 195 degrees of freedom
AIC: 191.73

Number of Fisher Scoring iterations: 5

> training.error.rate
[1] 0.22080
> test.error.rate
[1] 0.23054

10 Call:
glm(formula = V9 ~ V1 + V2 + V6 + V7, family = binomial)

Deviance Residuals:
Min	1Q	Median	3Q	Max
-2.3258	-0.6996	-0.3984	0.6484	2.1184

Coefficient	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-10.139008	1.530105	-6.626000	0.000000	***
V1	0.040410	0.059386	0.680000	0.496210	
V2	0.042816	0.007333	5.839000	0.000000	***
V6	0.113376	0.031906	3.553000	0.000380	***
V7	0.932747	0.551620	1.691000	0.090850	.

Signif. codes: 0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 263.58 on 199 degrees of freedom
Residual deviance: 181.57 on 195 degrees of freedom
AIC: 191.57

Number of Fisher Scoring iterations: 5

> training.error.rate
[1] 0.24000
> test.error.rate
[1] 0.23392

Method 3: Support Vector Machine:

For simplicity, we implement linear support vector machine without standardization of the inputs. We do not conduct model selection for SVM. We used a grid search to choose the tuning parameter C. There are 20 values to choose for C and the 20 values are 0.00125*2^i; i=0, 1, ..., 19.

Therefore, we do not fully take advantage of the predicting potential of the linear support vector machine.

1

```
&&&&The tuned C is: 0.08:

SVM solution:

The solution is:
-7.123441 0.031859 0.033056 -0.010631 0.026105 -0.002750 0.058901 1.144016
0.011771

Train error:0.23, Test error:0.227113.

 Number and Prop. of SVs: 105 0.525
```

2.

```
&&&&The tuned C is: 0.08:

SVM solution:


The solution is:
-5.120853 0.117058 0.029008 -0.015816 0.008251 0.001639 0.045930 0.199768 -
0.010009

Train error:0.235, Test error:0.230634.

 Number and Prop. of SVs: 109 0.545
```

3

```
&&&&The tuned C is: 0.01:

SVM solution:

The solution is:
-6.103253 0.092452 0.032367 -0.003491 -0.005098 -0.000965 0.035590 0.056580
0.016192

Train error:0.235, Test error:0.234155.

 Number and Prop. of SVs: 114 0.57
```

4

```
&&&&The tuned C is: 20.48:

SVM solution:
```

The solution is:
-6.848512 0.084157 0.035792 -0.032962 -0.011932 -0.000232 0.094851 0.636066
0.025615

Train error:0.23, Test error:0.228873.

 Number and Prop. of SVs: 105 0.525

5
&&&&The tuned C is: 0.64:

SVM solution:

The solution is:
-7.413804 0.099309 0.025589 0.012006 -0.017500 -0.001071 0.069950 0.541385
0.014887

Train error:0.2, Test error:0.234155.

 Number and Prop. of SVs: 96 0.48

6
&&&&The tuned C is: 2.56:

SVM solution:

The solution is:
-6.075539 0.007490 0.027956 -0.008279 0.029809 -0.001265 0.029704 0.229882
0.028475

Train error:0.28, Test error:0.211268.

 Number and Prop. of SVs: 129 0.645

7
&&&&The tuned C is: 0.16:

SVM solution:

The solution is:
-6.115617 0.181133 0.027831 -0.013500 0.000726 -0.000904 0.066840 0.713710 -
0.000904

Train error:0.19, Test error:0.235915.

 Number and Prop. of SVs: 97 0.485

8
&&&&The tuned C is: 0.16:

SVM solution:

The solution is:

-8.376623 0.047003 0.037932 -0.013615 -0.021416 -0.001288 0.108699 0.458968
0.021259

Train error:0.18, Test error:0.248239.

 Number and Prop. of SVs: 87 0.435

9

&&&&The tuned C is: 0.04:

SVM solution:

The solution is:
-6.805050 0.038360 0.032130 -0.005463 -0.025163 -0.001606 0.076611 0.280671
0.025093

Train error:0.2, Test error:0.234155.

 Number and Prop. of SVs: 96 0.48

10

&&&&The tuned C is: 0.04:

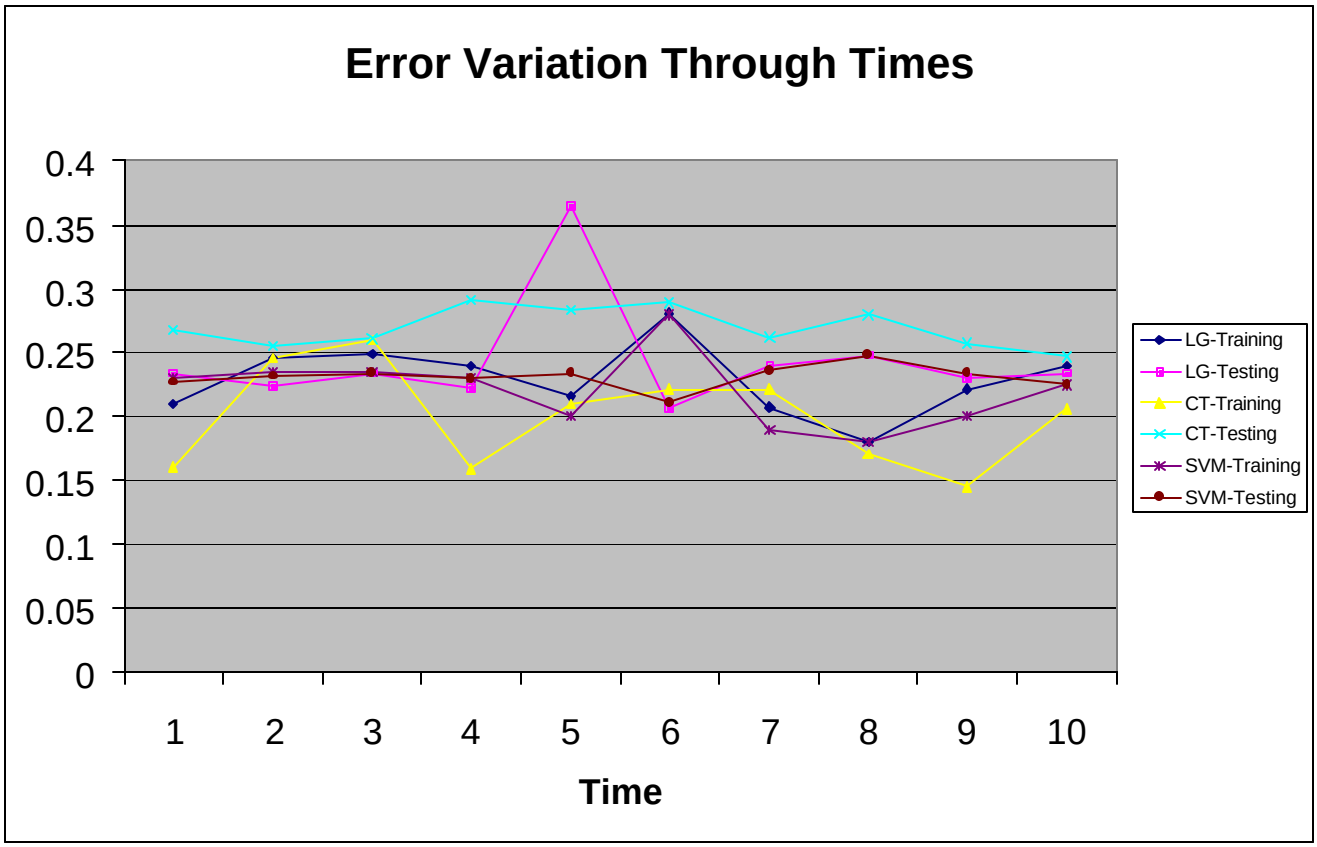
SVM solution:

The solution is:
-7.024884 -0.000675 0.027611 0.008831 0.038137 0.001051 0.030133 0.494360 -
0.001582

Train error:0.225, Test error:0.225352.

 Number and Prop. of SVs: 105 0.525

Comparison of these 3 methods:



Methods	Error	1	2	3	4	5	6	7	8	9	10	Ave.	Std.
Logistic	Training	0.2093	0.2458	0.2496	0.2400	0.2150	0.2803	0.2074	0.1805	0.2208	0.2400	0.2290	0.0280
Regression	Testing	0.2339	0.2238	0.2339	0.2217	0.3644	0.2075	0.2393	0.2481	0.2305	0.2339	0.2440	0.0438
Classification	Training	0.1599	0.2458	0.2592	0.1594	0.2093	0.2208	0.2208	0.1709	0.1450	0.2054	0.2000	0.0390
Tree	Testing	0.2677	0.2555	0.2603	0.2907	0.2833	0.2887	0.2623	0.2799	0.2569	0.2468	0.2690	0.0154
Support Vector Machine	Training	0.2300	0.2350	0.2350	0.2300	0.2000	0.2800	0.1900	0.1800	0.2000	0.2250	0.2210	0.0290
	Testing	0.2270	0.2310	0.2340	0.2290	0.2340	0.2110	0.2360	0.2480	0.2340	0.2250	0.2310	0.0094

The interesting discovery from the above table is that the smaller the training error is, the larger the testing error is. Support vector machine is the most powerful in prediction, since it has the smallest average test error rate and the smallest variance. The second one is logistic regression. And classification tree is the worst in predictability. As is pointed out at the beginning of the implementation of SVM, there are still enough margins to improve the performance of SVM. Therefore, SVM is the most promising method for this classification problem.

Appendix:

Code for nearest-neighbor imputation:

```
dbt<-read.table("C:/Documents and Settings/Administrator/My
Documents/Rfiles/diabetes_backup.txt")
attach(dbt)
data.dbt<-rep(0,6912)
dim(data.dbt)<-c(768,9)
data.dbt<-dbt

temp<-c(V1[V2!=0 & V3!=0 & V4!=0 & V5!=0 & V6!=0],V2[V2!=0 & V3!=0 & V4!=0 & V5!=0 &
V6!=0],V3[V2!=0 & V3!=0 & V4!=0 & V5!=0 & V6!=0],V4[V2!=0 & V3!=0 & V4!=0 & V5!=0 &
V6!=0],V5[V2!=0 & V3!=0 & V4!=0 & V5!=0 & V6!=0],V6[V2!=0 & V3!=0 & V4!=0 & V5!=0 &
V6!=0],V7[V2!=0 & V3!=0 & V4!=0 & V5!=0 & V6!=0],V8[V2!=0 & V3!=0 & V4!=0 & V5!=0 &
V6!=0],V9[V2!=0 & V3!=0 & V4!=0 & V5!=0 & V6!=0])

dim(temp)<-c(392,9)

for (i in 1:768)
{ if (data.dbt[i,2]==0 | data.dbt[i,3]==0 | data.dbt[i,4]==0 | data.dbt[i,5]==0 | data.dbt[i,6]==0)
{d<-1000000000000000
for (j in 1:392)
{ dn<-(data.dbt[i,1]-temp[j,1])^2
    for (k in 2:6)
          {if (data.dbt[i,k]!=0) dn<-dn+(data.dbt[i,k]-temp[j,k])^2
          }
     dn<-dn+(data.dbt[i,7]-temp[j,7])^2+ (data.dbt[i,8]-temp[j,8])^2
    dn<-sqrt(dn)
 if (dn<d)
 {d<-dn
  no<-j
 }
 }
 for (k in 2:6)
 { if (data.dbt[i, k]==0) data.dbt[i,k]<-temp[no,k]
 }
 }
 }
```

Code for Logistic Regression:

```
data.dbt<-read.table("C:/Documents and Settings/Administrator/My
Documents/Rfiles/NN_revised_exc0_no.txt")
attach(data.training)
dbt.lgt<-glm(V9~V1+V2+V6+V7, family=binomial)
summary(dbt.lgt)
dbt.fits<-fitted(dbt.lgt)
```

```
dbt.fits.class<-(dbt.fits>0.5)
dbt.training.class<-(V9==1)
dbt.training.comp<-dbt.training.class==dbt.fits.class
training.error.rate<-length(dbt.training.comp[dbt.training.comp==FALSE])/200
training.error.rate
data.test<-read.table("C:/Documents and Settings/Administrator/My Documents/Rfiles/tsset1.txt")

dbt.pred<-predict(dbt.lgt, data.frame(data.test), se.fit=F)
dbt.pred.class<-(dbt.pred>0)
dbt.test.class<-( data.test$V9==1)
dbt.test.comp<-dbt.pred.class==dbt.test.class
test.error.rate<-length(dbt.test.comp[dbt.test.comp==FALSE])/568
test.error.rate
```

Code for classification tree:

```
dbt.tree<-rpart(V9~V1+V2+V3+V4+V5+V6+V7+V8,data=data.training, method="class",
parms=list(split="information"))
printcp(dbt.tree)
plotcp(dbt.tree)

dbt.tree.pruned<-prune(dbt.tree, cp=0.046)
tree.fits<-predict(dbt.tree.pruned,data.frame(data.training))
tree.fits.class<-(tree.fits[,2]>0.5)
tree.training.class<-(V9==1)
tree.training.comp<-tree.training.class==tree.fits.class
tree.training.error.rate<-length(tree.training.comp[tree.training.comp==FALSE])/200
tree.training.error.rate

tree.pred<-predict(dbt.tree.pruned,data.frame(data.test))
tree.pred.class<-(tree.pred[,2]>0.5)
tree.test.class<-(data.test$V9==1)
tree.test.comp<-tree.test.class==tree.pred.class
tree.test.error.rate<-length(tree.test.comp[tree.test.comp==FALSE])/568
tree.test.error.rate

plot(dbt.tree.pruned,compress=T)
text(dbt.tree.pruned,use.n=T)
```

Code for support vector machine:

Yufeng Liu, Xiaotong Shen, and Hani Doss (2003) Multicategory Psi-Learning and Support Vector Machine: Computational Tools. Journal of Computational and Graphical Statistics, Accepted.

Reference:

Yufeng Liu, Xiaotong Shen, and Hani Doss (2003) Multicategory Psi-Learning and Support Vector Machine: Computational Tools. Journal of Computational and Graphical Statistics, Accepted.

Yi Lin, Yoonkyung Lee, Grace Wahba (2002) Support Vector Machines for Classification in Nonstandard Situations. Machine Learning, 46, 191-202, 2002.

John C. Platt (1999) Probalistic Outputs for Support vector machines and comparison to Regularized Likelihood Method. Advances in Large Margin Classifiers, A. Smola, P. Bartlett, B. Schölkopf, D. Schuurmans, eds., pp. 61-74, MIT Press, (1999).