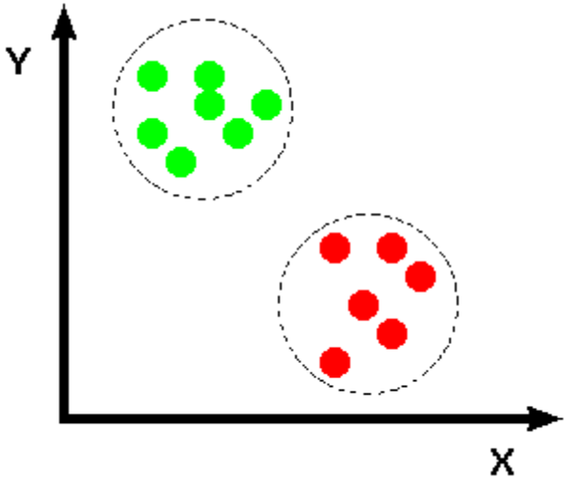
Advanced Analytics with R

Classification Methods

Classification

- ***Classification*** is about identifying to which of a set of categories a new observation belongs, on the basis of a training set of data.



Naïve Bayes Classification

Terms

- Classes – possible outcomes that we would classify an object to.
  - E.g. Buy or don't buy
  - Denoted as C_i
- Predictors – attribute data used to classify objects.
  - E.g. gender, income, marriage status, education background
  - Denoted as $x_1, x_2, \dots, x_p$

Cut-off Probability

- In general, classification is about finding $P(C_i|x_1, x_2, \dots, x_p)$.
- A cut-off probability *p* is pre-defined.
- If $P(C_i|x_1, x_2, \dots, x_p)$>p, then the object will be classified under class C_i.
- When p=0.5, the classification rule is called majority voting rule.

Exact Bayes Procedure

- $P\left(C_i \middle| x_1, \ldots, x_p\right) = \frac{P(x_1,\ldots,x_p|C_i)P(C_i)}{P\left(x_1, \ldots, x_p \middle| C_1\right)P(C_1)+\cdots+P(x_1,\ldots,x_p|C_m)P(C_m)}$
- However, when the number of predictors gets larger, many of the records to be classified will be without exact matches.
- For example, a male loan applicant, age 45, married with 5 children, graduated from Techno India University, manager. Is he risky or safe customer?

Naïve Bayes

- $$P\left(C_i \middle| x_1, \dots, x_p\right) = \frac{[P(x_1 \mid C_i)P(x_2 \mid C_i)\dots P\left(x_p \middle| C_i\right)]P(C_i)}{[P(x_1 \mid C_1)P(x_2 \mid C_1)\dots P\left(x_p \middle| C_1\right)]P(C_1)+\dots+[P(x_1 \mid C_m)P(x_2 \mid C_m)\dots P\left(x_p \middle| C_m\right)]P(C_m)}$$

Lasagna Triers Example

- Suppose we are going to use the following predictors to classify customers: Age, PayType, Nbhd.
- The probability of a customer with Age = 30, PayType = “Hourly” and Nbhd = “West” is then

$$P(Yes \mid Age = 30, PayType = \text{"Hourly"}, \text{Nbhd="West"})$$

$$= \frac{P(Age = 30|Yes)P(PayType = \text{Hourly}|Yes)P(Nbhd = \text{West}|Yes)P(Yes)}{P(Age = 30|Yes)P(PayType = \text{Hourly}|Yes)P(Nbhd = \text{West}|Yes)P(Yes) + P(Age = 30|No)P(PayType = \text{Hourly}|No)P(Nbhd = \text{West}|No)P(No)}$$

Categorical Predictor

- P(Yes)

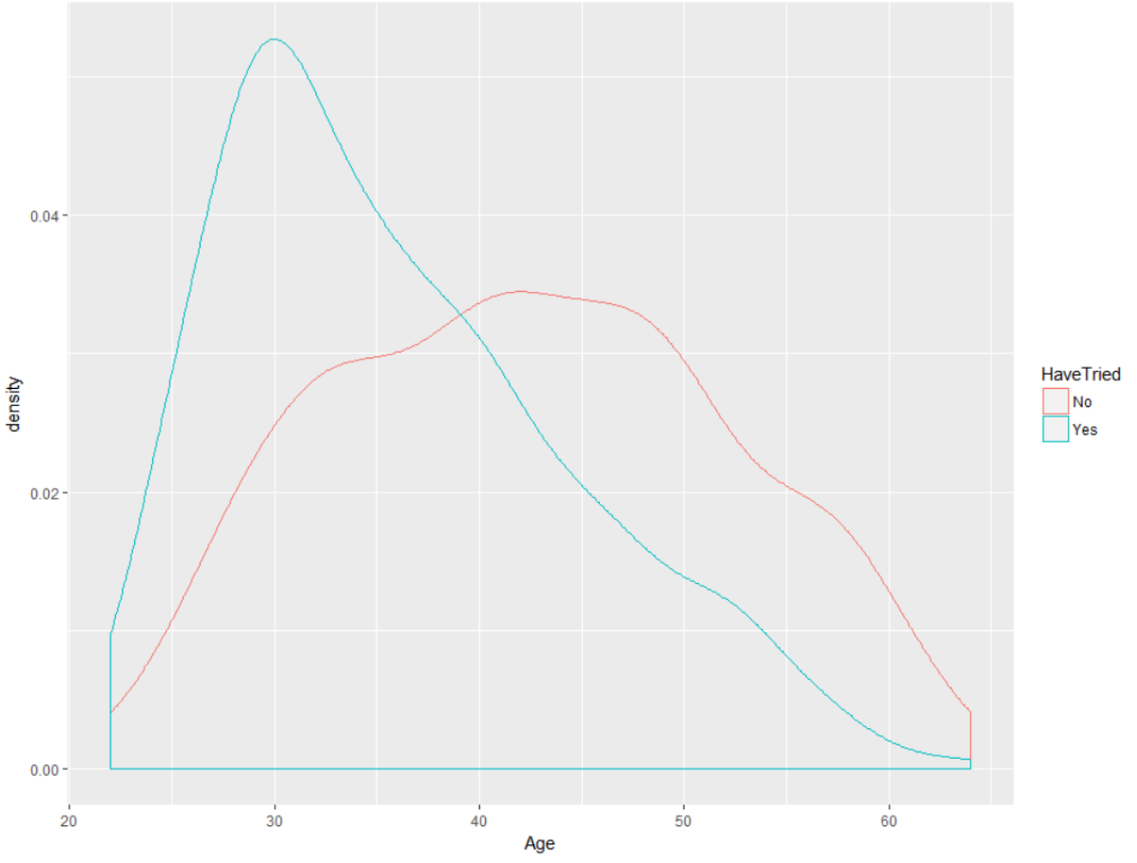
```
nrow(triers.train[triers.train$HaveTried=="Yes",])/nrow(triers.train)
```

- P(Nbhd="West" | Yes)

```
nrow(triers.train[triers.train$HaveTried=="Yes"&triers.train$Nbhd=="West",])/nrow(triers.train
[triers.train$HaveTried=="Yes",])
```

Continuous Predictor

- Using the attributes' mean and standard deviation, for each of the two class outcomes (HaveTried=Yes and HaveTried=No), we estimate the ***pdf*** value at any given decision point and use it as the conditional probability in the formula.



Example on Age

- Mean & Standard Deviation of Age among triers

```
mean.triers <- mean(triers.train[triers.train$HaveTried=="Yes",]$Age)

sd.triers <- sd(triers.train[triers.train$HaveTried=="Yes",]$Age)
```

- Mean & Standard Deviation of Age among non-triers

```
mean.nontriers <- mean(triers.train[triers.train$HaveTried=="No",]$Age)
sd.nontriers <- sd(triers.train[triers.train$HaveTried=="No",]$Age)
```

- P(Age=30 | Yes)

```
dnorm(30,mean.triers,sd.triers)
```

Use e1071 package

- Model

```
m <- naiveBayes(HaveTried ~ ., data = triers.train)
```

- Prediction

```
predict(m, triers.train)
```

- Evaluation

```
a<-table(predict(m, triers.train), triers.train$HaveTried)
accuracy <- (a[1,1]+a[2,2])/sum(a)
accurary
```

Decision Tree Classification

German Credit Case

- File: germancredit.csv
- The data set contains attributes and outcomes on 1000 loan applications.
- To construct a model to predict if a loan applicant is going to default.

Splitting

- Choose the predictor variable that best splits the data into two groups such that the ***purity*** (homogeneity) of the outcome in the two groups is maximized (that is, as many default cases in one group and non-default cases in the other group as possible).
- If the predictor is categorical, combine the categories to obtain two groups with maximum purity.
- If the predictor is continuous, choose a cut-off point that maximizes purity for the two groups created.

Comparison of different splits

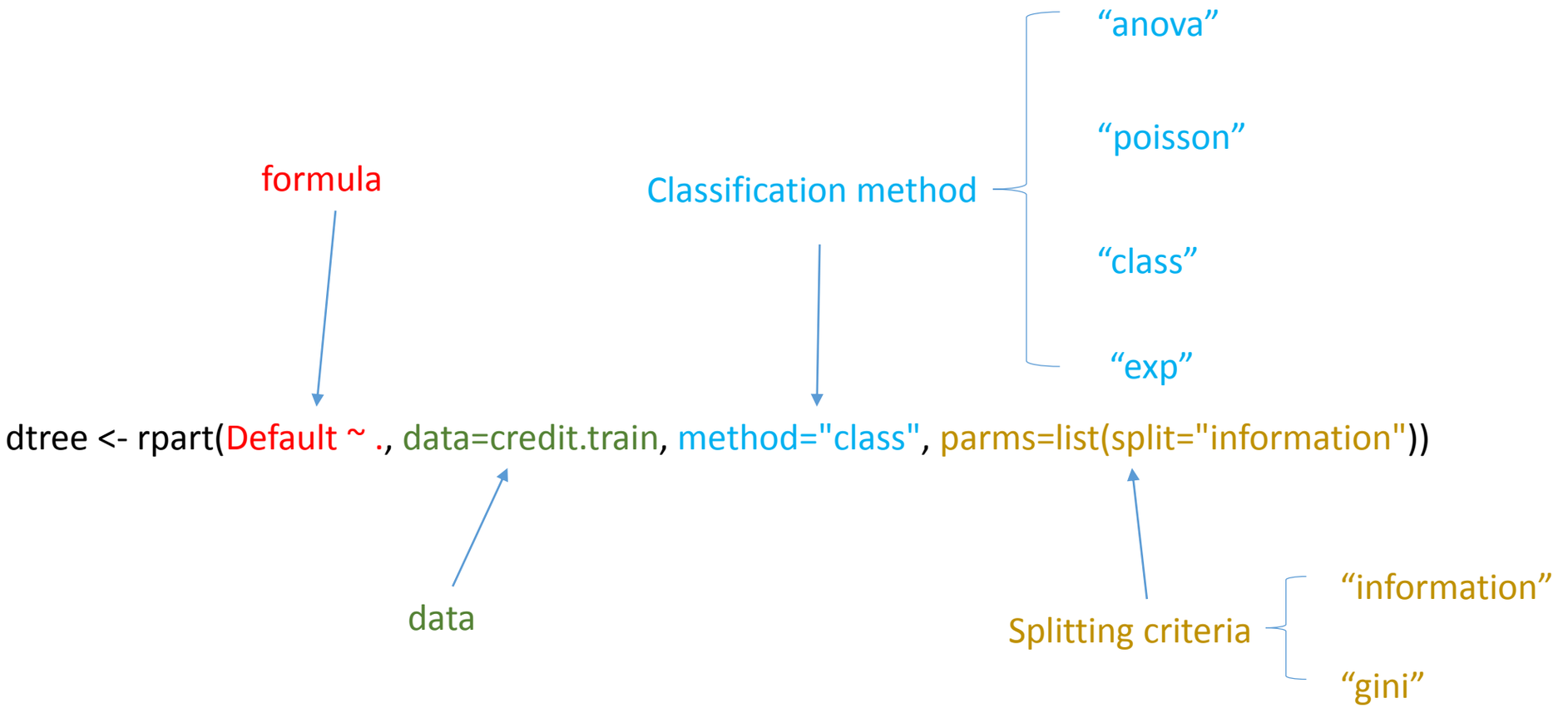
```
> df.foreign
           0   1 default.rate
foreign 667 296    0.3073728
german   33   4    0.1081081
```

```
> df.history
              0   1 default.rate
good         36  53    0.5955056
poor        421 197    0.3187702
terrible    243  50    0.1706485
```

Decision Tree Procedure

- Step 1: Split data by a predictor
- Step 2: Separate the data into these two groups, and continue the process for each sub-group.
- Step 3: Repeat Step 1 and 2 until a subgroup contains fewer than a minimum number of observations or no splits decrease the impurity beyond a specific threshold.

Model


formula
Classification method
“anova”
“poisson”
“class”
“exp”
dtree <- rpart(Default ~ ., data=credit.train, method="class", parms=list(split="information"))
data
Splitting criteria
“information”
“gini”

Summary

the error rate for a tree of a given size

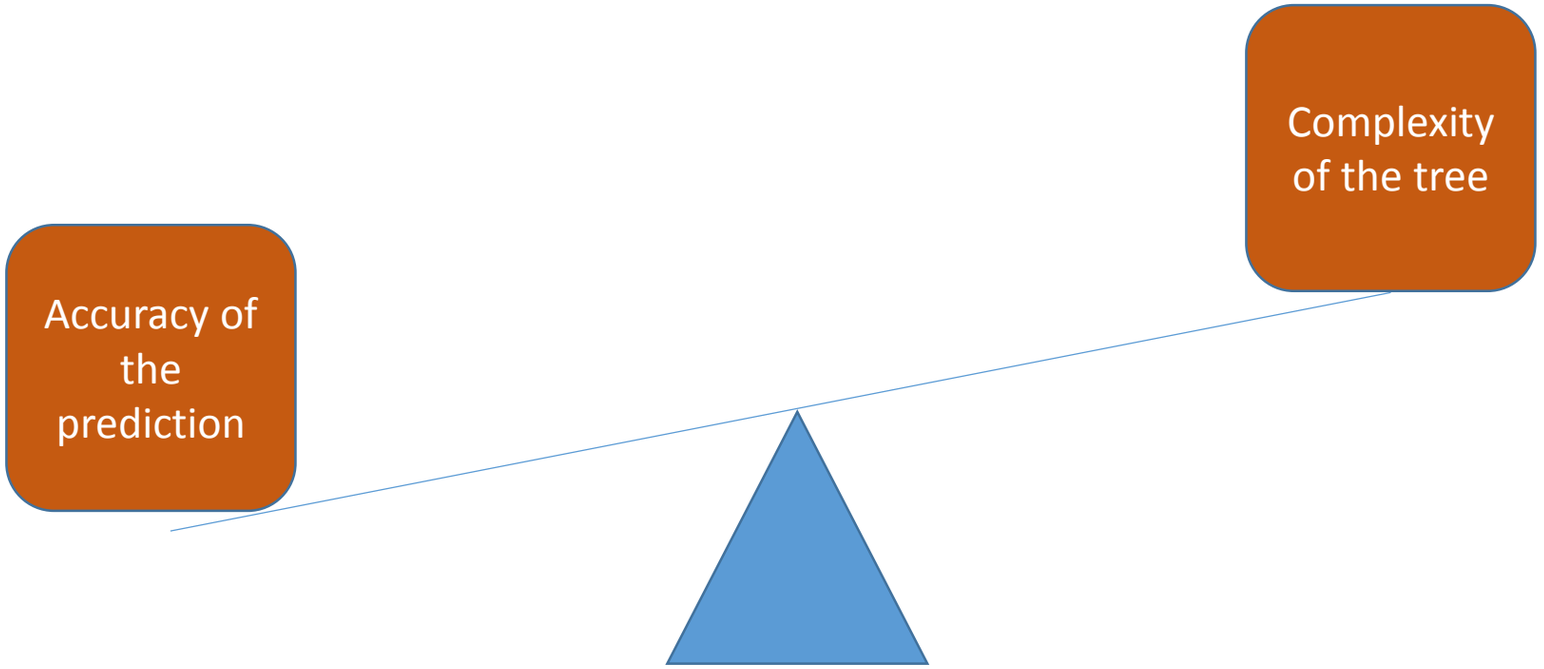
the standard error of the cross-validation error

```
          CP nsplit rel error    xerror       xstd
1 0.04424779      0 1.0000000 1.0000000 0.05623302
2 0.03539823      3 0.8407080 1.0663717 0.05729455
3 0.01991150      4 0.8053097 1.0088496 0.05638141
4 0.01327434      6 0.7654867 0.9203540 0.05479830
5 0.01032448     10 0.7123894 0.9026549 0.05445437
6 0.01000000     13 0.6814159 0.9070796 0.05454123
```

Complexity Parameter

cross validated error based on 10-fold cross validation

Trade-off in Decision Tree



cp

- Internally, rpart keeps track of something called the complexity of a tree.
- The complexity measure is a combination of the size of a tree and the ability of the tree to separate the classes of the target variable. If the next best split in growing a tree does not reduce the tree's overall complexity by a certain amount, rpart will terminate the growing process.
- This amount is specified by the ***complexity parameter***, cp, in the call to rpart.
- Setting cp to a negative amount ensures that the tree will be fully grown.

Other means of controlling growth of the tree

- ***minsplit***: the minimum number of observations that must exist in a node in order for a split to be attempted.
- ***minbucket***: the minimum number of observations in any terminal node.
- ***maxcompete***: the number of competitor splits retained in the output
- ***maxsurrogate***: the number of surrogate splits retained in the output.
- ***Usesurrogate***: how to use surrogates in the splitting process. It takes options 0, 1, or 2.
- ***xval***: number of cross-validations.
- ***surrogatestyle***: controls the selection of a best surrogate. Options: 1 and 2.
- ***maxdepth***: Set the maximum depth of any node of the final tree, with the root node counted as depth 0.

Pruning Tree – Method 1

	CP	nsplit	rel error	xerror	xstd
1	0.04424779	0	1.0000000	1.0000000	0.05623302
2	0.03539823	3	0.8407080	1.0663717	0.05729455
3	0.01991150	4	0.8053097	1.0088496	0.05638141
4	0.01327434	6	0.7654867	0.9203540	0.05479830
5	0.01032448	10	0.7123894	0.9026549	0.05445437
6	0.01000000	13	0.6814159	0.9070796	0.05454123

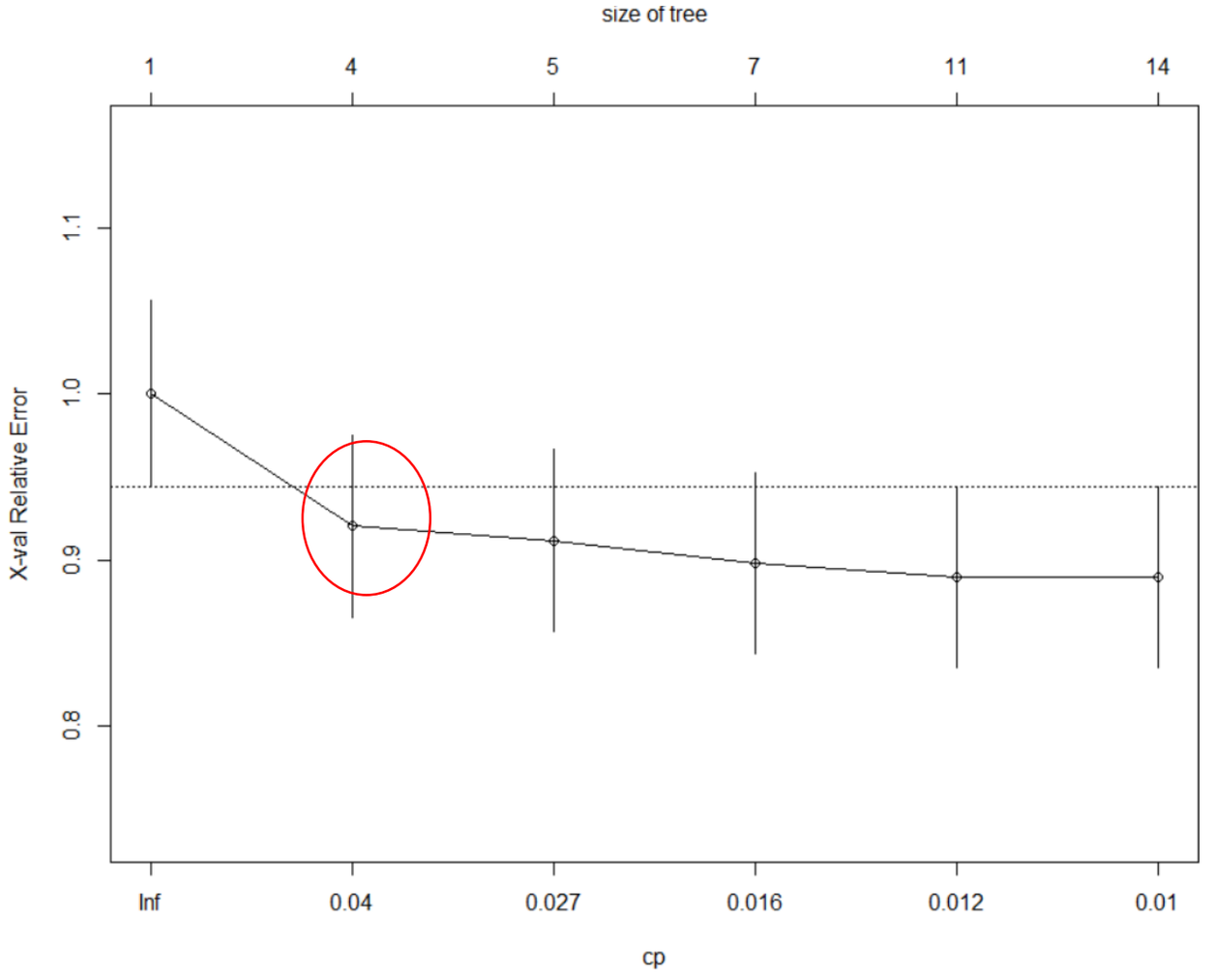
Choose the smallest tree whose cross-validated error is within one standard error of the minimum cross-validated error value.

The threshold is 0.9070796 + 0.05454123

The smallest tree is Node 2 with nsplit = 3 and the corresponding cp value to use is some value above 0.03539823, for example 0.04

Pruning Tree - Method 2

```
plotcp(dtree)
```



Choose the tree size associated with the largest cp below the dotted line.

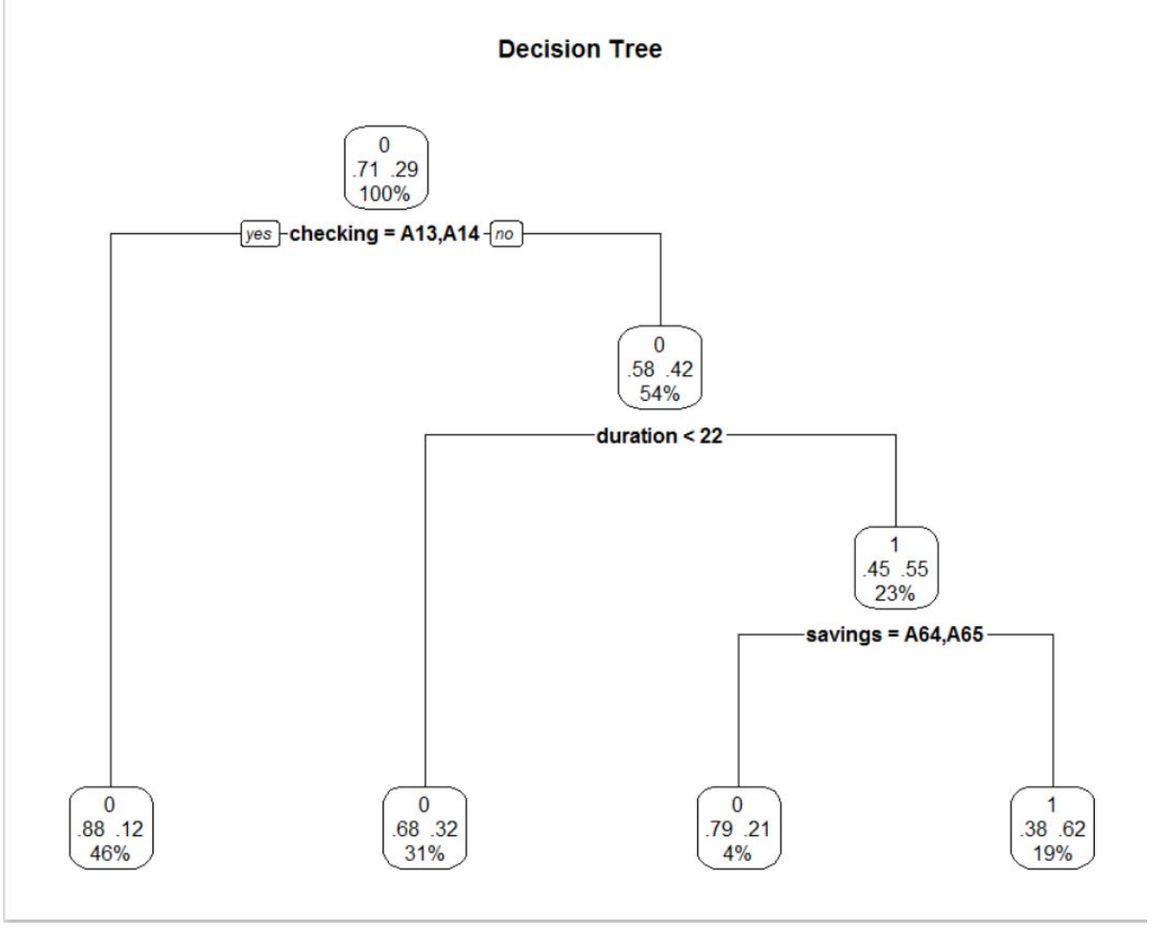
Prune the tree

dtree.pruned <- prune(dtree, cp=0.04)

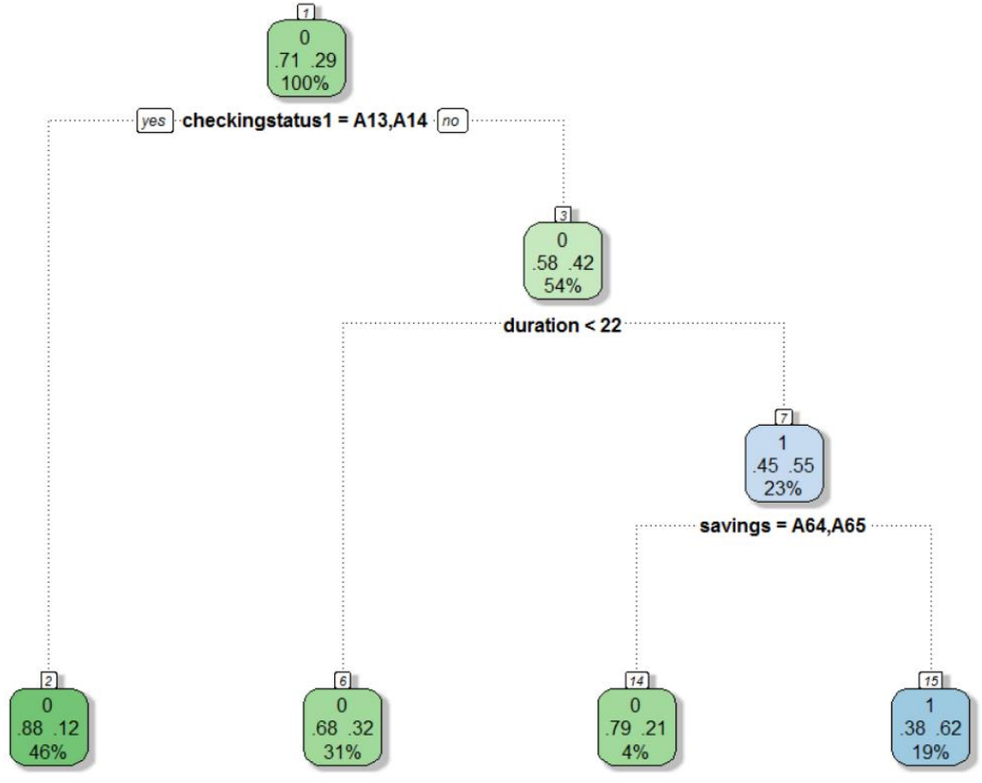
```
> dtree.pruned$cptable
          CP nsplit rel error   xerror       xstd
1 0.04424779      0  1.000000 1.000000 0.05623302
2 0.04000000      3  0.840708 0.920354 0.05479830
```

Visualizing Tree

```
prp(dtree.pruned, type=2, fallen.leaves = TRUE,
extra=104, main="Decision Tree")
```




```
Require(rattle)
fancyRpartPlot(dtree.pruned)
```



Measuring the prediction outcomes

```
dtree.pred <- predict(dtree.pruned, credit.test, type = "response")

dtree.pred <- predict(dtree.pruned, credit.test, type = "class")


dtree.perf <- table (credit.test$Default, dtree.pred,dnn=c("Actual","Predicted"))
```

```
> dtree.perf
      Predicted
Actual   0   1
     0 119  15
     1  47  27
```

Cost Matrix

What if the cost of a default case is five times the benefit of a non-default case?

	Predicted	
Actual	Non-Default	Default
Non-Default	0	1
Default	5	0

```
dtree <- rpart(Default ~ ., data=credit.train, method="class",
parms=list(split="information",loss=matrix(c(0,1,5,0),
byrow=TRUE, nrow=2)))
```

Conditional Inference Trees

Algorithm

- Calculate p-values for the relationship between each predicator and the outcome variable.
- Select the predictor with the lowest p-value
- Explore all possible binary splits on the chosen predictor and dependent variable, and pick the most significant split.
- Separate the data into these two groups, and continue the process for each subgroup
- Continue until splits are no longer significant or the minimum node size is reached.

ctree() function

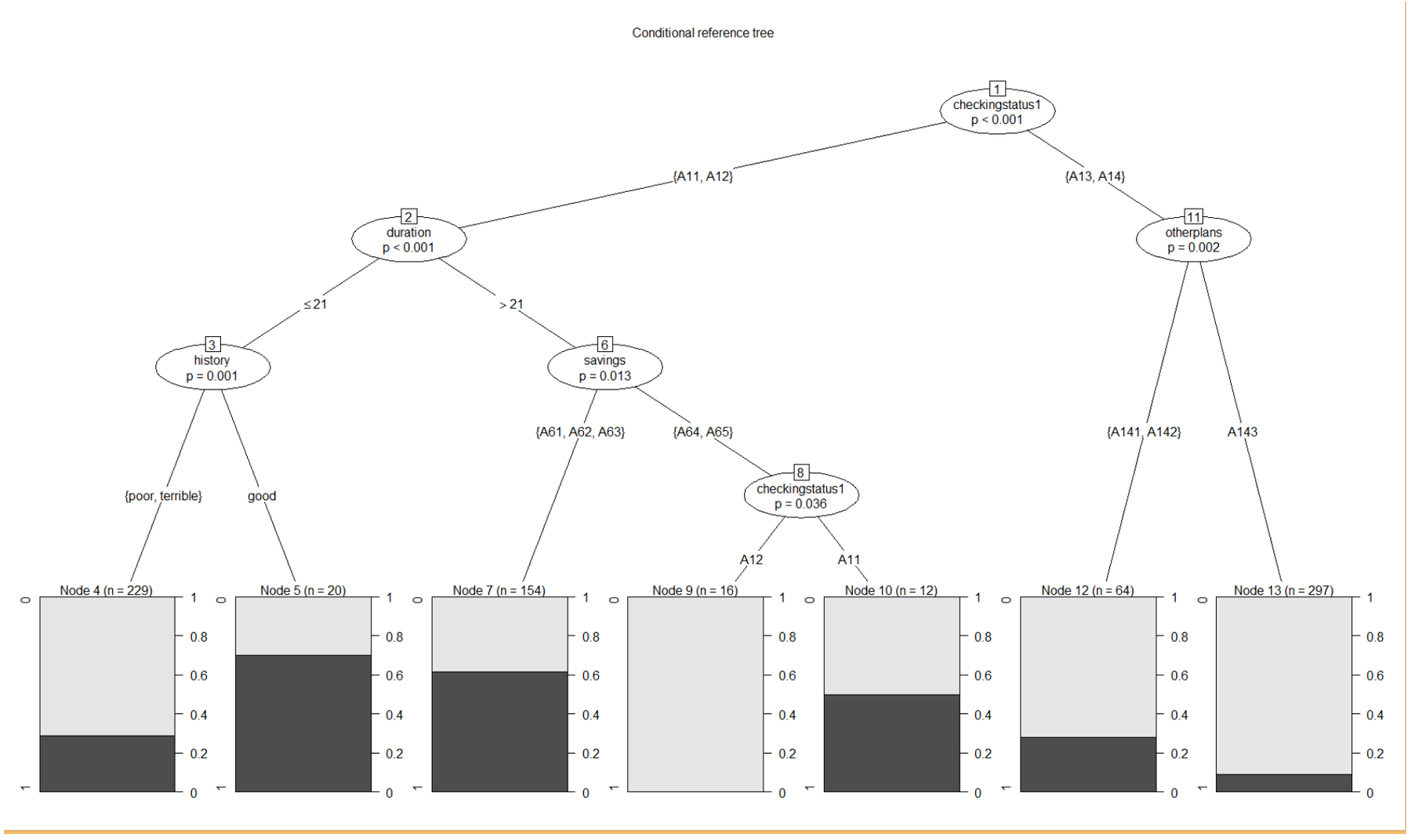
```
library(party)

fit.ctree <- ctree(Default ~ ., credit.train)

ctree.predict <- predict(fit.ctree, credit.test, type="response")

ctree.perf <- table(credit.test$Default, ctree.predict, dnn=c("Actual","Predicted"))
```

```
> ctree.perf
      Predicted
Actual   0   1
     0 118  16
     1  40  34
```

```
> dtree.perf
      Predicted
Actual   0   1
     0 119  15
     1  47  27
```

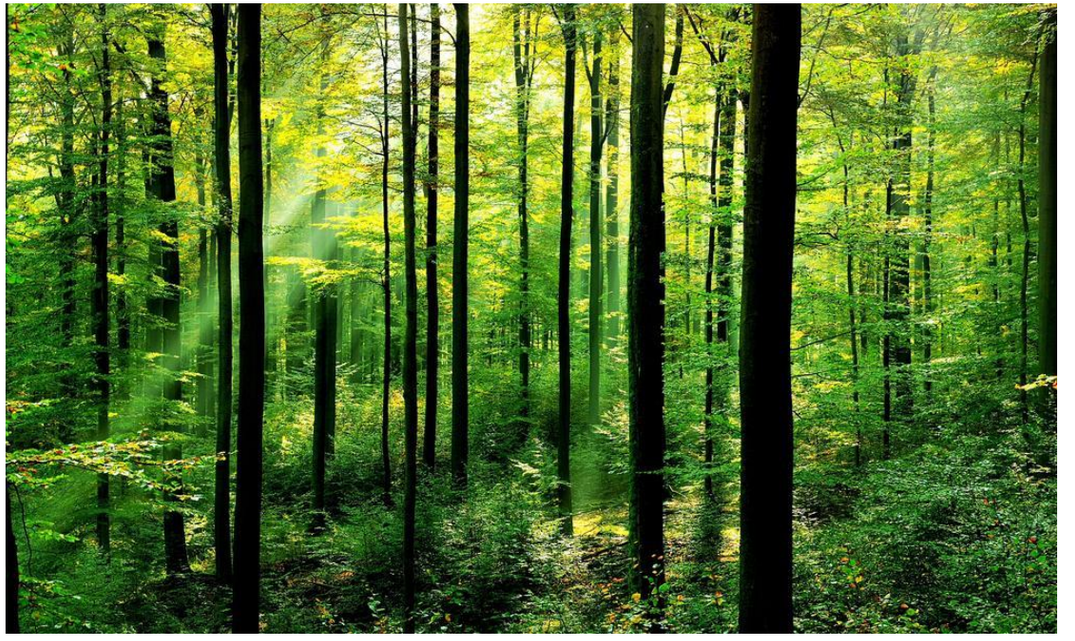
Ctree view

```
plot(fit.ctree,main="Conditional reference tree")
```



Adding Weights

```
fit.ctree<- ctree(Default~., credit.train, weights= ifelse(credit.train$Default==1, 5, 1))
```



Random Forest

Algorithm

- Grow a large number of decision trees by sampling N cases with replacement from the training set.
- Sample m<N variables at each node. These variables are considered candidates for splitting in that node. The value *m* is the same for each node.
- Grow each tree fully without pruning.
- Terminal nodes are assigned to a class based on the mode of cases in that node.
- Classify new cases by sending them down all the trees and taking a vote – majority rules.

Model

```
library(randomForest)

set.seed(1234)

fit.forest <- randomForest(Default ~., data=credit.train, na.action = na.roughfix, importance=TRUE)
```

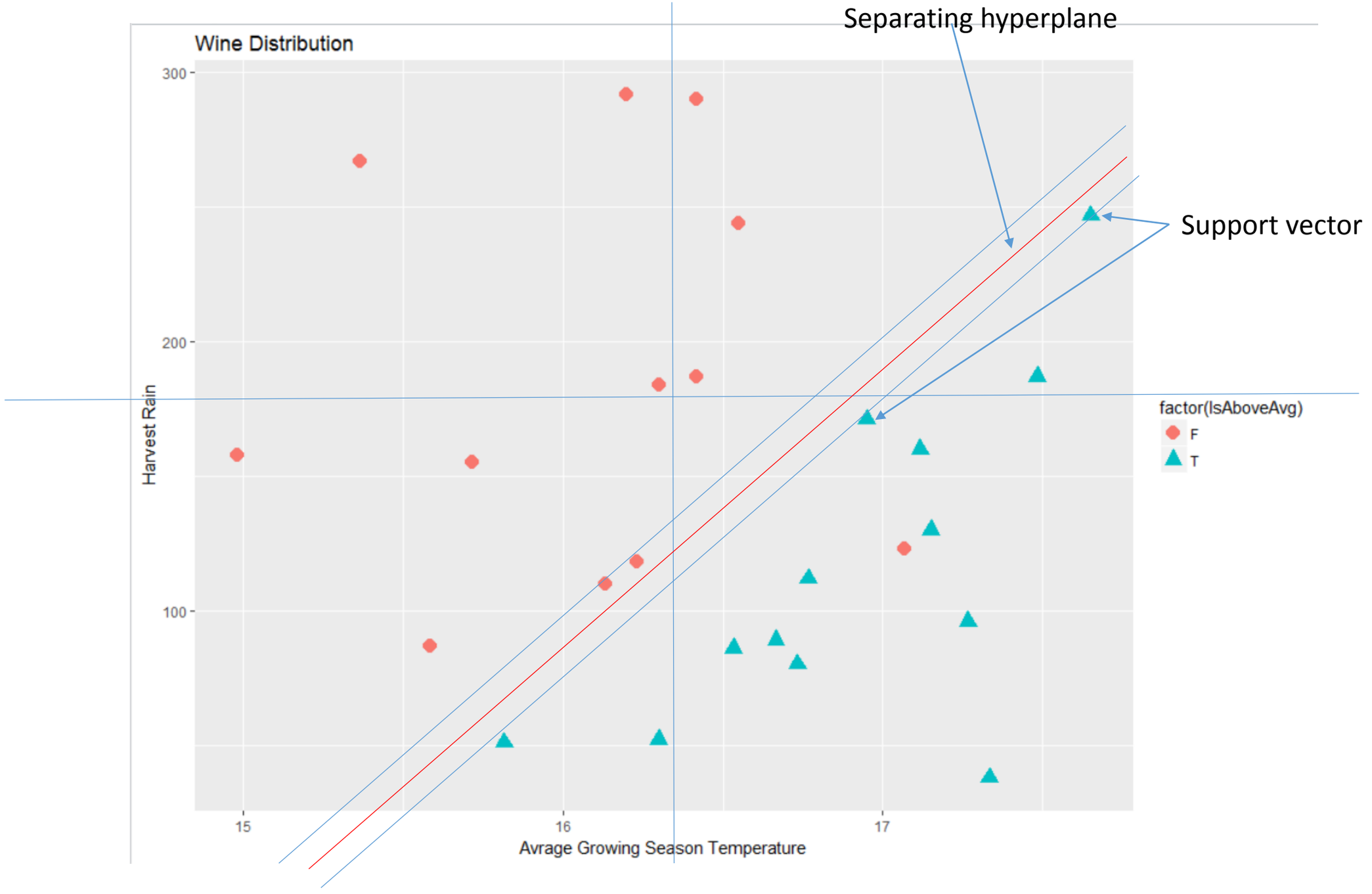
Take cost into consideration

```
fit.forest <- randomForest(Default ~., data=credit.train, na.action = na.roughfix,
importance=TRUE, cutoff=c(5/6,1/6))
```

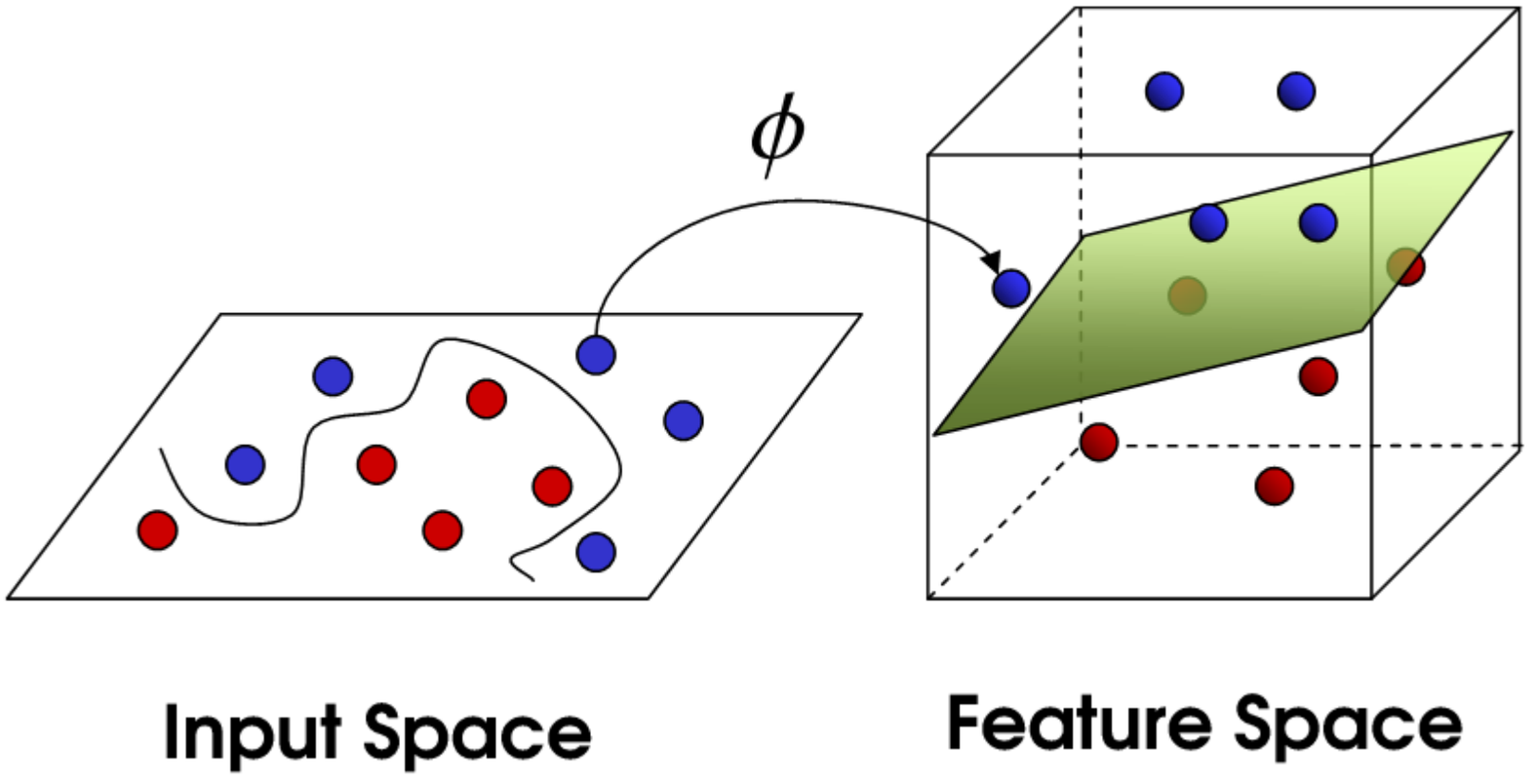
Support Vector Machines

Support Vector Machines (SVM)

- SVM seeks an optimal hyperplane for separating two classes in a multidimensional space.
- The hyperplane is chosen to maximize the margin between the two classes' closest points.
- The points on the boundary of the margin are called ***support vectors***.
- The middle of the margin is the ***separating hyperplane***.


Separating hyperplane
Support vector
Wine Distribution
300
200
100
Harvest Rain
15
16
17
Avrage Growing Season Temperature
factor(IsAboveAvg)
F
T

Some data that is complex in lower dimension becomes simple in higher dimension



Model

- library(e1071)
- fit.svm <- svm(Default ~., data=credit.train)
- fit.svm
- svm.pred <- predict (fit.svm, credit.train)

Tuning performance

- Two parameters ***gamma*** and ***cost*** are affecting the performance of SCM.
- Larger values of gamma result in a larger number of support vectors.
- The cost parameter represents the cost of making errors. A large value severely penalizes errors and leads to a more complex classification boundary. It may lead to overfitting problem.

Tuning Performance

```
tuned <- tune.svm(Default ~., data=credit.train, gamma=10^(-6:1),cost=10^(-10:10))
```

```
tuned
```

Handling asymmetric cost

```
credit.train$Default <- factor(credit.train$Default, levels=c(0,1), labels=c("NDF","DF"))
```

```
fit.svm.cost <- svm(Default ~., data=credit.train, class.weights=c(NDF=1,DF=5))
```