the data resources:

http://archive.ics.uci.edu/ml/datasets/Adult

incomedata<-read.csv("adult.csv",header=TRUE)

attach(incomedata)

age[which(age=="?")]

workclass[which(workclass=="?")]<-NA

levels(education)

levels(marital.status)

levels(occupation)

occupation[which(occupation=="?")]<-NA

levels(relationship)

levels(race)

levels(sex)

class(education.num)

capital.gain[which(capital.gain=="?")]

capital.loss[which(capital.loss=="?")]

hours.per.week[which(hours.per.week=="?")]

levels(native.country)

native.country[which(native.country=="?")]<-NA

incomedata$income<-factor(incomedata$income,levels=c("<=50K",">50K"),labels=c("0","1"))))

the above is to check whether there is missing value. and because in this document missing value is "?",so we replace it by NA.

install.packages("caret")

library(caret)

install.packages("randomForest")

library(randomForest)

library(rpart)

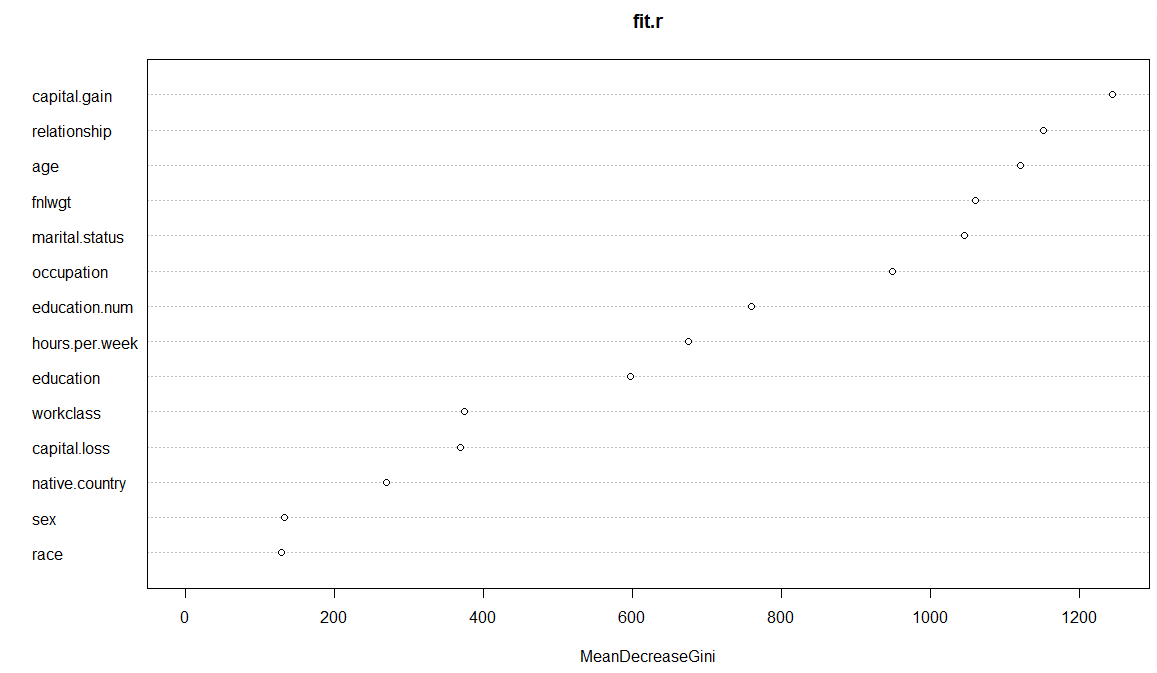
to install several packages

#dimension reduction

fit.r<-randomForest(factor(income)~., data=incomedata)

varImp(fit.r)

varImpPlot(fit.r,type=2)



to use random forest to reduce the variable selection bias. because for every node it is only random choose features for classification case that use to pick up a splitting rule. and the overall performance will be better.

and use variable importance to see the plot to know the sequence of importance.

my.control<-rpart.control(cp=0,xval=10)

fit.p<-rpart(income~.,data=incomedata,method="class",control=my.control)

printcp(fit.p)

use rpart function to get a classification result. and prune it by using 1+SE rule.

1+SE rule is designed to keep the tree as simple as possible without sacrificing much accuracy

and to reduce instability in tree selection

result:

Classification tree:

rpart(formula = income ~ ., data = incomedata, method = "class",

control = my.control)

Variables actually used in tree construction:

[1] age capital.gain capital.loss education education.num

[6] fnlwgt hours.per.week marital.status native.country occupation

[11] race relationship sex workclass

Root node error: 7841/32561 = 0.24081

n= 32561

CP nsplit rel error xerror xstd

1 1.2639e-01 0 1.00000 1.00000 0.0098399

2 6.4022e-02 2 0.74723 0.74723 0.0088402

3 3.7495e-02 3 0.68320 0.68320 0.0085321

4 4.8463e-03 4 0.64571 0.64571 0.0083394

5 4.5913e-03 11 0.60133 0.60451 0.0081162

6 3.8260e-03 12 0.59674 0.59686 0.0080734

7 3.2521e-03 13 0.59291 0.59316 0.0080525

8 1.7855e-03 16 0.58169 0.58832 0.0080250

9 1.6580e-03 21 0.57123 0.58220 0.0079900

10 1.5304e-03 23 0.56791 0.58283 0.0079937

11 1.3179e-03 25 0.56485 0.58079 0.0079820

12 1.2753e-03 28 0.56090 0.57965 0.0079753

13 1.1478e-03 30 0.55835 0.57709 0.0079606 ##smallest testing error

14 8.9274e-04 31 0.55720 0.57990 0.0079768

15 8.2898e-04 32 0.55631 0.58003 0.0079776

16 7.0144e-04 41 0.54789 0.57926 0.0079731

17 6.6318e-04 43 0.54649 0.58105 0.0079834

18 6.3767e-04 53 0.53807 0.58003 0.0079776

19 5.7391e-04 62 0.53169 0.58181 0.0079878

20 5.4202e-04 74 0.52404 0.57965 0.0079753

21 5.1014e-04 82 0.51970 0.57965 0.0079753

22 4.4637e-04 88 0.51664 0.57850 0.0079687

23 4.2512e-04 92 0.51486 0.58003 0.0079776

24 4.1449e-04 98 0.51231 0.58220 0.0079900

25 3.8260e-04 104 0.50950 0.58220 0.0079900

26 3.4009e-04 132 0.49828 0.58245 0.0079915

27 3.1884e-04 135 0.49726 0.58538 0.0080083

28 2.9758e-04 149 0.49190 0.58615 0.0080127

29 2.5507e-04 159 0.48871 0.59559 0.0080662

30 2.4232e-04 169 0.48616 0.59814 0.0080806

31 2.2319e-04 237 0.46002 0.60094 0.0080963

32 2.1256e-04 241 0.45913 0.60388 0.0081127

33 2.0406e-04 280 0.44828 0.60605 0.0081248

34 1.9130e-04 285 0.44726 0.60872 0.0081396

35 1.7005e-04 327 0.43719 0.61115 0.0081530

36 1.5942e-04 344 0.43426 0.61574 0.0081783

37 1.5304e-04 349 0.43336 0.61638 0.0081818

38 1.4575e-04 355 0.43234 0.61714 0.0081860

39 1.2753e-04 362 0.43132 0.62824 0.0082463

40 1.0932e-04 443 0.42035 0.63066 0.0082593

41 1.0203e-04 457 0.41882 0.63487 0.0082819

42 9.5651e-05 462 0.41831 0.63487 0.0082819

43 8.5023e-05 497 0.41474 0.63959 0.0083070

44 6.3767e-05 513 0.41309 0.64137 0.0083165

45 5.4658e-05 558 0.41015 0.65158 0.0083702

46 5.1014e-05 565 0.40977 0.65387 0.0083822

47 4.2512e-05 585 0.40862 0.65668 0.0083968

48 3.1884e-05 603 0.40786 0.65859 0.0084067

49 2.5507e-05 607 0.40773 0.66127 0.0084205

50 1.8219e-05 631 0.40709 0.66509 0.0084402

51 1.4171e-05 638 0.40696 0.66560 0.0084428

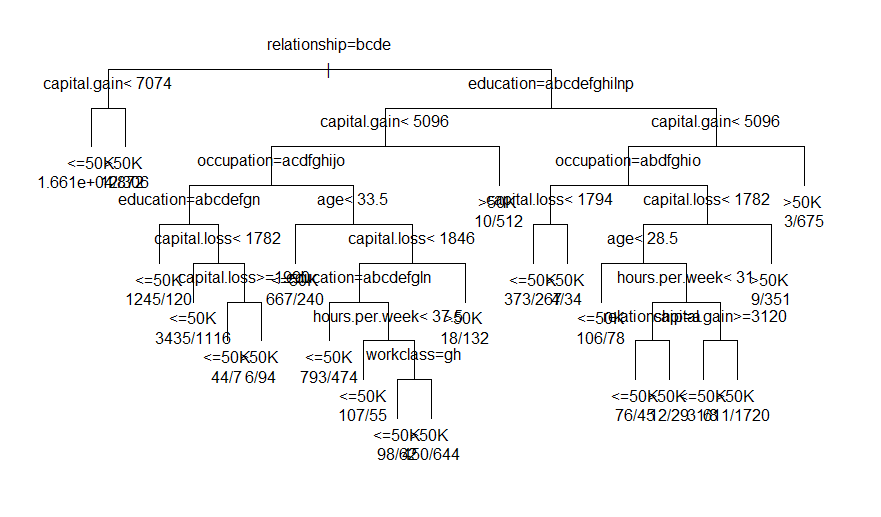
52 0.0000e+00 647 0.40684 0.66611 0.0084455

0.57709+0.0079606=0.5850506

tree9<-prune(fit.p,cp=(1.7855e-03 +1.6580e-03)/2)

plot(tree9,uniform = T,margin=0.2)

text(tree9,use.n = T)



pred9<-predict(tree9,incomedata,type="class")

table(incomedata$income,pred9)

result:

pred9

<=50K >50K

<=50K 23585 1135

>50K 3344 4497

error9<-table(incomedata$income,pred9)[1,2]+table(incomedata$income,pred9)[2,1]

error.rate<-error9/length(income)

> error.rate

[1] 0.1375572

#because in the raw data, the data are unbalanced, 25% classified by 0, 75%classified by 1.So if we use priors to modify it ,we could get more accurate predict results when we want to balance the misclassification cases between two categories.

fit.pp<-rpart(income~.,data=incomedata,parms=list(prior=c(0.25,0.75)), method="class",control=my.control)

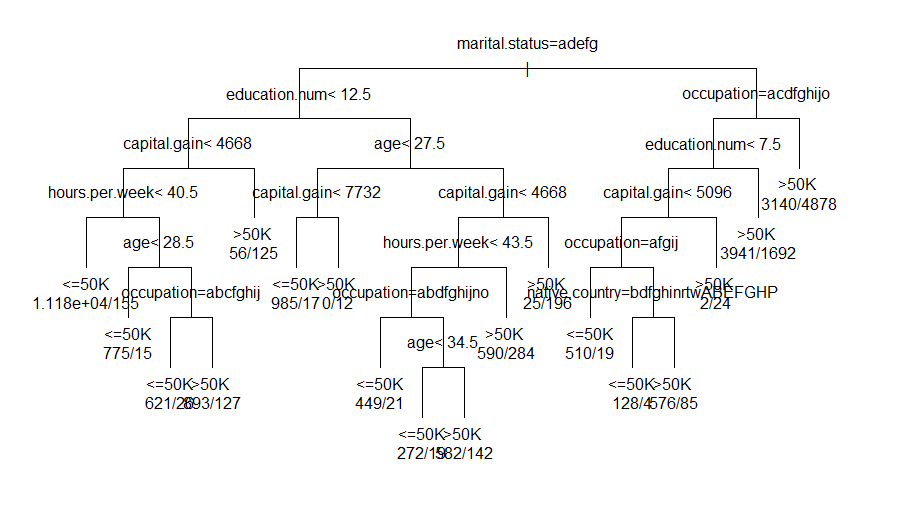
printcp(fit.pp)

0.50509+0.0080517

prior.tree9<-prune(fit.pp,cp=(2.2083e-03+2.5963e-03)/2)

plot(prior.tree9,uniform = T,margin=0.2)

text(prior.tree9,use.n = T)



prior.pred9<-predict(prior.tree9,incomedata,type="class")

table(incomedata$income,prior.pred9)

prior.pred9

<=50K >50K

<=50K 14915 9805

>50K 276 7565

prior.error9<-table(incomedata$income,prior.pred9)[1,2]+table(incomedata$income,prior.pred9)[2,1]

prior.error.rate<-prior.error9/length(income)

> prior.error.rate

[1] 0.3096035

we could see that the >50K misclassified cases reduce and the <=50K misclassified cases increase. It depends on which goal you want to get and which mistake you weight more(which mistake you don't want to see it happens)