Stat 2025 - HW 8

Michael Discenza

April 8, 2013

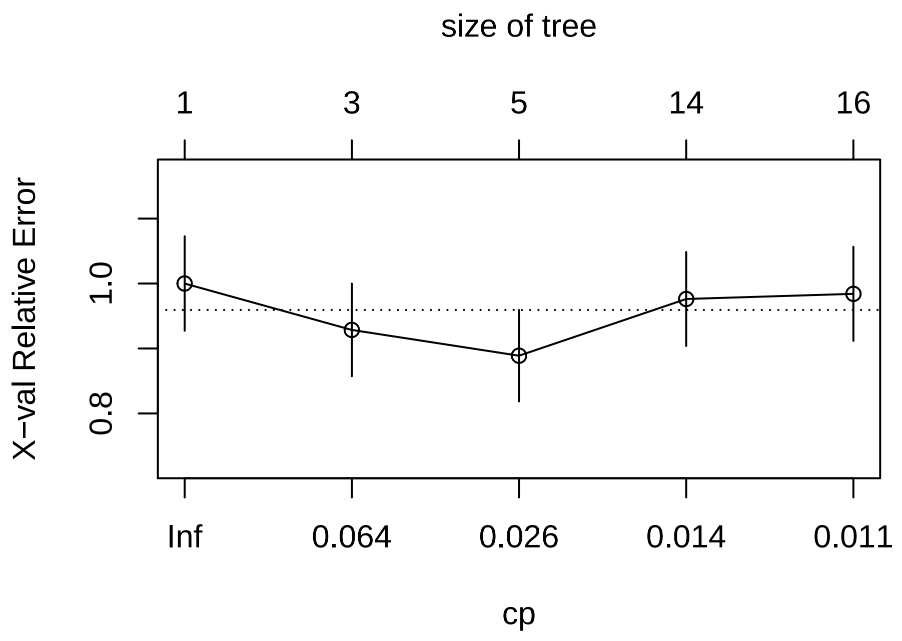
Setting up data

```r
SAH <- read.table("http://stat.columbia.edu/~madigan/W2025/data/SAHmissing.txt",
    header = TRUE, sep = "\t")
SAH <- na.exclude(SAH)  # only use the records for which we have the response variable
validation.cases <- sample(1:dim(SAH)[1], 42, replace = F)  # randomly select 42 records to hold out
validation.data <- SAH[validation.cases, ]
training.data <- SAH[-validation.cases, ]
```

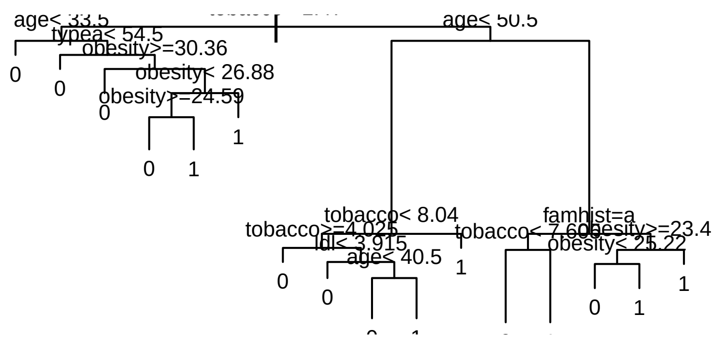
Fitting a Classification Tree Model

First we fit a full single tree. We look at the plot of cross validation errors against the various complexity parameters, which correspond to trees of different sizes

```r
library(rpart)
tree1 <- rpart(chd ~ ., data = training.data, method = "class")
plotcp(tree1)
```



```
plot(tree1)
text(tree1, cex = 0.7)
```



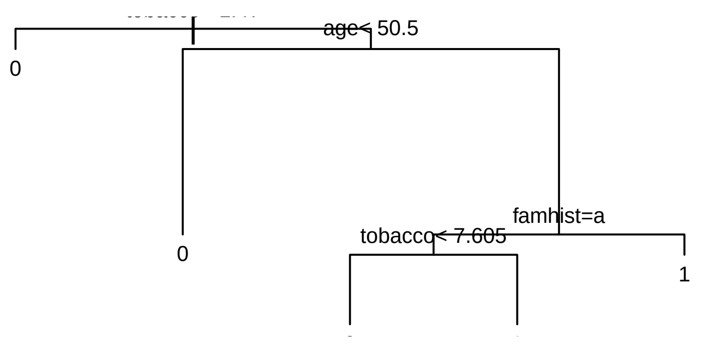
```
# optimal cp:
tree1$cptable[which.min(tree1$cptable[, "xerror"]), "CP"]

## [1] 0.01587
```

We see that we obtain the best cross validation error by fitting a tree with a complexity parameter of `0.01968504`. Thus we prune the tree to that level. We also try prunning the tree to a smaller tree to try to compensate for over fitting. In this case, we have to us a complexity parameter of 0.026, where, we can see form the above CP plot, that the tree size drops from 6 to 5. effective when we predict. In the validation phase of this exercise, we can see which tree performs better on the validation set.

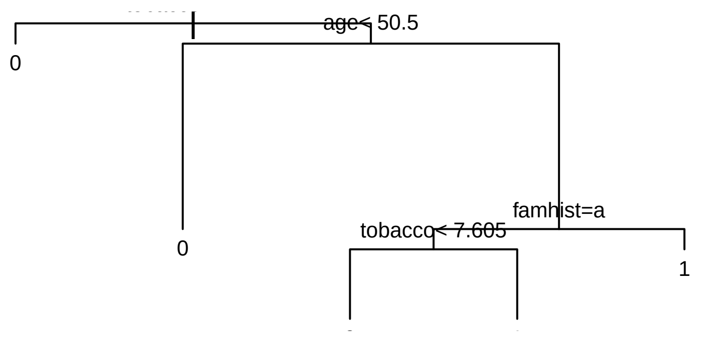
```
# selecting the optimum cp to prune the tree
tree1.prunned1 <- prune(tree1, cp = tree1$cptable[which.min(tree1$cptable[,
    "xerror"]), "CP"])
plot(tree1.prunned1, main = "tree1.prunned1 (CP =0.019")
text(tree1.prunned1, cex = 0.7)
```

tree1.prunned1 (CP =0.019



```r
tree1.prunned2 <- prune(tree1, cp = 0.026)
plot(tree1.prunned2, main = "tree1.prunned2 (CP=0.026")
text(tree1.prunned2, cex = 0.7)
```

tree1.prunned2 (CP=0.026



Fitting a Random Forest Model

To make this exercise slightly more interesting we can look and see how a random forest model performs on this data set, relative to a single tree.

```
library(randomForest)

## randomForest 4.6-7
## Type rfNews() to see new features/changes/bug fixes.

rf <- randomForest(y = as.factor(training.data$chd), x = training.data[, -10],
    ntree = 100, type = "class", replace = TRUE)
print(rf)

##
## Call:
##  randomForest(x = training.data[, -10], y = as.factor(training.data$chd),      ntree = 100, replace =
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 31.22%
## Confusion matrix:
##     0  1 class.error
## 0 212 40      0.1587
## 1  78 48      0.6190
```

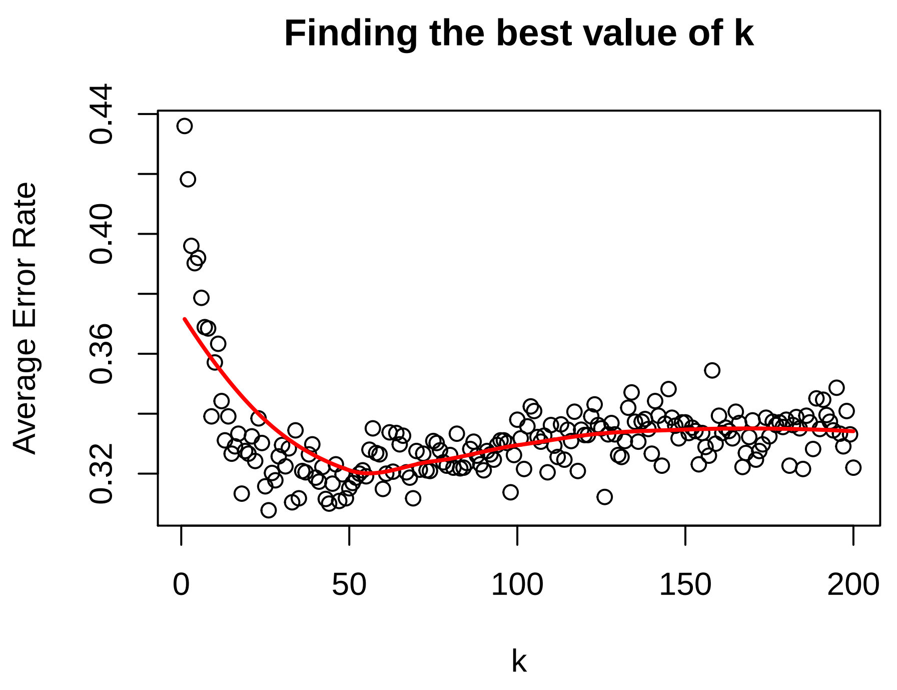
Predicton using Knn

The first task in using knn is to determine the optimal k. To find the best k, we can only use the training data and not the validation data, because in the latter scenario, we would be introducing leakage to the model fitting. As a result, we must subest the training data in order to evaluate the effectiveness of certain k values. The code below test gets the average of the misclassification error for 100 different test samples of size 50 withheld from the training data for each value of k from 1 to 200.

```
library(class)
training.data2 <- training.data
validation.data2 <- validation.data
# We transform this factor variable to a numberic variable so that KNN
# works with the dataframe
training.data2$famhist <- as.numeric(as.factor(training.data$famhist))
validation.data2$famhist <- as.numeric(as.factor(validation.data$famhist))

err <- NULL
k <- NULL
for (i in 1:200) {
    err.temp <- NULL
    for (j in 1:100) {
        kfitting.cases <- sample(1:dim(training.data2)[1], 45, replace = F)  # randomly select 42 recor
        kvalidation.data <- training.data2[kfitting.cases, ]
        ktraining.data <- training.data2[-kfitting.cases, ]
        knn.response <- knn(train = ktraining.data[, -10], test = kvalidation.data[,
            -10], cl = ktraining.data[, 10], k = i, l = 0, use.all = TRUE)
        err.temp[j] <- sum(abs(as.numeric(as.character(knn.response)) - kvalidation.data[,
            10]))/dim(kvalidation.data)[1]
        # print(err.temp[j])
    }
    # print(err.temp)
    err[i] <- mean(err.temp)
    k[i] <- i
}
k.values <- as.data.frame(cbind(k, err))
colnames(k.values) <- c("k", "err")


y.loess <- loess(y ~ x, span = 0.75, data.frame(x = k, y = err))
y.predict <- predict(y.loess, data.frame(x = k))
plot(k, err, main = "Finding the best value of k", ylab = "Average Error Rate")
lines(k, y.predict, col = "red", lwd = 2)
```



```
y.fitted <- unlist(y.predict)
kmin = y.predict[which.min(y.predict)]
kmin

##     56
## 0.3201
```

From the above graph, we can see that a reasonable choice for k would be a value slightly larger than 50. The minimum fitted value from the loess curve (span=0.75) that we can fit to the average errors confirms that the best value for k is 56. Thus we proceed to use that value when we apply the knn algorithm.

```
# calculating the fitted values
knn.response <- knn(train = training.data2[, -10], test = validation.data2[,
    -10], cl = training.data2[, 10], k = 56, l = 0, use.all = TRUE)
err.knn <- sum(abs(as.numeric(as.character(knn.response)) - validation.data2[,
    10]))/dim(validation.data2)[1]
err.knn
```

```
## [1] 0.3571
```

Prediction Using Other Models (Logistic Regression and GAM)

We can also compare these results to logistic regression:

```
logreg <- glm(chd ~ ., data = training.data, family = "binomial")
log.reg.pred <- predict.glm(logreg, newdata = validation.data[, -10], type = "response")
err.rate.log.reg <- sum(abs(round(log.reg.pred) - validation.data[, 10]))/dim(validation.data)[1]
```

And we can also fit a GAM for comparison:

```
library(mgcv)

## This is mgcv 1.7-21.  For overview type 'help("mgcv-package")'.

library(mgcv)
library(boot)
gam1 <- gam(chd ~ s(sbp, bs = "cs") + s(tobacco, bs = "cs") + s(ldl, bs = "cs") +
    s(adiposity, bs = "cs") + s(typea, bs = "cs") + s(obesity, bs = "cs") +
    famhist + s(alcohol, bs = "cs") + s(age, bs = "cs"), data = training.data,
    family = binomial)
gam.pred <- predict.gam(gam1, newdata = validation.data[, -10], type = "response")
err.rate.gam <- sum(abs(round(gam.pred) - validation.data[, 10]))/dim(validation.data)[1]
```

Comparing results

First, we define a function that caclculates misclassification error for our model of interests:

```
mis_class_err <- function(model, validation.data = validation.data) {
    if (class(model) == "randomForest") {
        response <- round(as.numeric(as.character(predict(model, newdata = validation.data))))
        misclasserr <- sum(abs(response - validation.data[, 10]))/dim(validation.data)[1]
    } else {
        response <- round(predict(model, newdata = validation.data)[, 2])
        misclasserr <- sum(abs(response - validation.data[, 10]))/dim(validation.data)[1]
    }
    return(misclasserr)
}
```

Then we we combine the the results from each model into a single data frame:

```
models <- NULL
err <- NULL
# tree1
models[1] <- "Unprunned Tree"
err[1] <- mis_class_err(tree1, validation.data)
models[2] <- "Prunned tree (cp=0.197)"
err[2] <- mis_class_err(tree1.prunned1, validation.data)
models[3] <- "Prunned tree (cp=0.260)"
err[3] <- mis_class_err(tree1.prunned2, validation.data)
```

```r
models[4] <- "Random Forest"
err[4] <- mis_class_err(rf, validation.data)
models[5] <- "Knn"
err[5] <- err.knn
models[6] <- "Regular Logistic Regression (full model)"
err[6] <- err.rate.log.reg
models[7] <- "Logistic GAM (CS smoothed full model)"
err[7] <- err.rate.gam

results <- cbind(models, err)
```

A table of our final results shows that the knn perfomed best on the held-out data followed by the prunned tree:

	models	err
1	Unprunned Tree	0.285714285714286
2	Prunned tree (cp=0.197)	0.261904761904762
3	Prunned tree (cp=0.260)	0.238095238095238
4	Random Forest	0.30952380952381
5	Knn	0.19047619047619
6	Regular Logistic Regression (full model)	0.285714285714286
7	Logistic GAM (CS smoothed full model)	0.261904761904762

Discussion:

Looking at these results, we can see that the first tree, the unprunned tree likely overfit the training data so did not perform as well on the validationd data. Both of prunned tree models perform well, beating the regular logistic model and tying with the GAM. The smaller model (higher CP) ends up performing better, probably because the larger model overfit the training data, like the unprunned tree.

The random forest model (with 100 trees) did not perform particualrly well in the trial shown here, but in another trial, it only classified one record wrong (at a rate of 0.03), which indicates one of the problems with evaluating model performance on such a small held-out subset.

I am not exactly sure why that was the case, but k nearest neighboors performs very well on this data.

Finally, we see that the parametric logistic regression model, which has many useful properties that allow interpetation of the model, does not perform as well based only on predictive performance.

To illustrate the instability of these misclassification error rates and their dependency on the sample of the 42 records that are withheld, below, are the results for each of the models for a different 42 records:

	models	err
1	Unprunned Tree	0.333333333333333
2	Prunned tree (cp=0.197)	0.261904761904762
3	Prunned tree (cp=0.260)	0.261904761904762
4	Random Forest	0.30952380952381
5	Knn	0.19047619047619
6	Regular Logistic Regression (full model)	0.261904761904762
7	Logistic GAM (CS smoothed full model)	0.404761904761905