Integrating the code and text

A, A, & R

24 April, 2017

Contents

General set up for Aim 2b:

We assume we have a training and a test set. From here forward the training set, $(Z_i, W_i), i = 1 \dots n$, will be used for model building and the test set will solely be used for prediction error once we have a final model. Starting with all of the training data, under L_2 loss, both *partDSA* and CART would ordinarily seek to minimize

$$\min_{c_L} \sum_i I\{W_i \in Q_L(j,s)\}(Z_i - c_L)^2 + \min_{c_R} \sum_i I\{W_i \in Q_R(j,s)\}(Z_i - c_R)^2 \quad (1)$$

over all variables j and split points s, where $Q_L(j,s) = \{W|W_j \leq s\}$ and $Q_R(j,s) = \{W|W_j > s\}$. The (j,s) combination minimizing (1) can be determined quickly; in CART, the "best" choice is used to divide the root node into two daughter nodes and the above splitting process is then repeated within each daughter node. *partDSA* proceeds similarly, making use of (1) in the addition and substitution steps.

Here, we assume that our training set has been split into two independent subsets: a learning set, $(Z_{0i}, W_{0i}), i = 1 \dots n_0$; and, an evaluation set, $(Z_{1i}, W_{1i}), i = 1 \dots n_1$. We first apply an aggregate learner (e.g., $partDSA_{RF}$) to the learning set $(Z_{0i}, W_{0i}), i = 1 \dots n_0$, generating the (black box) prediction rule, $\hat{m}(w)$; then, we calculate predicted outcomes based on the covariates in the evaluation set, $\widehat{Z}_{1i} = \hat{m}(W_{1i}), i = 1 \dots n_1$. Importantly: $\hat{m}(w)$ and $(Z_{1i}, W_{1i}), i = 1 \dots n_1$ can be considered independent; in addition, the predictions $\widehat{Z}_{1i}, i = 1 \dots n_1$, only utilize the W_{1i}'s and not the Z_{1i}'s. Importantly, $\widehat{Z}_{1i}, i = 1 \dots n_1$, will not be used as covariates but rather in creating a target for shrinkage, thereby reducing variance.

Strategies 1 & 2 are discussed in the grant application and represent methods for modifying the loss function (1) used for split determination, by making use of the $\widehat{Z}_{1i}$'s to help guide splitting decisions. Here we provide further details that underpin Strategy 2.

Strategy 2

In addition to $\widehat{m}(\cdot)$, the ensemble building process provides (i) a measure of prediction error, $\hat{\sigma}_0^2$, derived from the learning set, $(Z_{0i}, W_{0i}), i = 1 \dots n_0$; and, (ii) B predicted outcomes for each evaluation set W_{1i}, generating both a predicted mean, $\widehat{Z}_{1i}$, and measure of variance, $\hat{\gamma}_i$. Strategy 2 leverages this information through penalization; the crux of this proposal involves replacing, for $v \in \{L, R\}$, the two optimization problems in (1) with

$$\min_{c_v} \textstyle\sum_i I\{W_{1i} \in Q_v(j,s)\} \left[(Z_{1i} - c_v)^2 + \lambda \alpha_i (c_v - \widehat{Z}_{1i})^2\right] \tag{2}$$

for $\alpha_i > 0$, each having the weighted-average solution

$$\widehat{c}_{v,j,s}(\lambda) = r_{v,j,s}(\lambda)\bar{Z}_{1v}(j,s) + (1 - r_{v,j,s}(\lambda))\hat{\bar{Z}}_{1v}(j,s), \tag{3}$$

where:

$$r_{v,j,s}(\lambda) = 1/(1 + \lambda \bar{\alpha}_{v,j,s}), \tag{4}$$

$$\bar{Z}_{1v}(j,s) = n_{v,j,s}^{-1} \textstyle\sum_i I(W_{1i} \in Q_v(j,s)) Z_{1i}, \tag{5}$$

$$\hat{\bar{Z}}_{1v}(j,s) = \{\textstyle\sum_i I(W_{1i} \in Q_v(j,s))\alpha_i \hat{Z}_{1i}\}/\{\textstyle\sum_i I(W_{1i} \in Q_v(j,s))\alpha_i\} \tag{6}$$

and

$$\bar{\alpha}_{v,j,s} = n_{v,j,s}^{-1} \sum_i I(W_{1i} \in Q_v(j,s))\alpha_i \tag{7}$$

where $n_{v,j,s} = \sum_i I(W_{1i} \in Q_v(j,s))$.

The choice $\alpha_i^{-1} = \text{var}(\hat{Z}_{1i})$ is probably best from an efficiency perpsective; hence the choice of $\alpha_i = \hat{\gamma}_i^{-1}$ in the grant application text. Note that, given $\bar{\alpha}_{j,v,s}$, more weight is placed on $\hat{\bar{Z}}_{1v}(j,s)$ when λ is larger; similarly, given λ, more weight is placed on $\hat{\bar{Z}}_{1v}(j,s)$ when $\bar{\alpha}_{j,v,s}$ is larger. Both make sense in shrinking the parameter estimate towards this weighted mean function.\

To decide:

1. Is this the most appropriate formulation?
2. How to choose λ.

Choice of λ: Within-Node

Applying the results of Section and assuming all expectation calculations are conditional on $W_{1i}, i \geq 1$ it can be shown that

$$K_2 = n_{v,j,s}^{-1}$$

and

$$K_1 = \hat{\bar{Z}}_{1v}(j,s).$$

Assuming that $E(Z_{1i}) = \mu_{Z_1}$ and $var(Z_{1i}) = \sigma^2_{Z_1}$ when $I(W_{1i} \in Q_v(j,s)) = 1$ (i.e., constant mean and variance within a node), the "best" within-node choice of λ via (19) becomes

$$\lambda_{opt} = \frac{n^{-1}_{v,j,s}\sigma^2_{Z_1}}{\bar{\alpha}_{v,j,s}(\mu_{Z_1} - \hat{\bar{Z}}_{1v}(j,s))^2}. \tag{8}$$

Note that selecting

$$\hat{\bar{Z}}_{1v}(j,s) = \{\sum_i I(W_{1i} \in Q_v(j,s))\hat{Z}_{1i}\}/\{\sum_i I(W_{1i} \in Q_v(j,s))\} \tag{9}$$

in equation (8) instead of $\hat{\bar{Z}}_{1v}(j,s)$ (defined in (6)) gives an alternative shrinkage target. There are other choices as well. Thus, Strategy 2 can be viewed as a procedure for shrinking the node-specific estimates towards some node-specific average predicted value.

Choice of λ: Prediction Error with Grid

Instead of using a within-node selection of λ, we can implement a global method for picking λ by minimizing the prediction error over a grid of possible values for λ. Suppose that $\hat{\mathcal{M}}(W,\lambda)$ denotes the final prediction rule obtained using the data (Z,W) – meaning, this is obtained from our proposed penalized loss procedure for fixed λ. Let $\mathcal{N}_1(\lambda),\ldots,\mathcal{N}_{K(\lambda)}(\lambda)$ be the partitions obtained in the final structure built with fixed λ; then, we know

$$\hat{\mathcal{M}}(W_{1i},\lambda) = \sum_{k=1}^{K(\lambda)} I\{W_{1i} \in \mathcal{N}_k(\lambda)\}\hat{c}_k(\lambda) \tag{10}$$

(piecewise constant predictor within each partition/node). Here,

$$\hat{c}_k(\lambda) = r_k(\lambda)\bar{Z}_{1j} + (1 - r_k(\lambda))\hat{\bar{Z}}_{1k}$$

where $r_k(\lambda) = 1/(1 + \lambda\bar{\alpha}_k(\lambda))$, $\bar{Z}_{1k}$ is the node-specific mean of the Z_{1i}s,

$$\hat{\bar{Z}}_{1k} = \{\sum_i I(W_{1i} \in \mathcal{N}_k(\lambda)\alpha_i\widehat{Z}_{1i}\}/\{\sum_i I(W_{1i} \in \mathcal{N}_k(\lambda))\alpha_i\}$$

and

$$\bar{\alpha}_k(\lambda) = \{\sum_i I(W_{1i} \in \mathcal{N}_k(\lambda))\alpha_i\}/\{\sum_i I(W_{1i} \in \mathcal{N}_k(\lambda))\}.$$

This is a very complicated function of λ and the within-node procedure described in Section probably cannot be directly adapted to choose a global λ.

Per Efron & Tibshirani (1993) and Efron (2004),

$$\text{err}(\lambda) := \sum_{i=1}^{n_1}(Z_{1i} - \hat{\mathcal{M}}(W_{1i},\lambda))^2 \tag{11}$$

is a version of the "apparent" prediction error because $\hat{\mathcal{M}}(W_{1i},\lambda)$ is built using the data(Z,W). As this is an optimistic assessment of error, we do not want to use it to choose λ. Following Efron (2004) a preferred measure of error is

$$\text{Err}(\lambda) := E_{Z_{20},W_{20}}\left[(Z_{20} - \hat{\mathcal{M}}(W_{20},\lambda))^2\right]$$

where (Z_{20}, W_{20}) is independent of $(Z_{1i}, W_{1i}), i = 1 \dots n_1$ and $\hat{\mathcal{M}}(w, \lambda)$ is held fixed in the expectation calculation. Calculations in Efron (2004, Eqn. 2.8) show

$$E[\text{Err}(\lambda)] = E[\text{err}(\lambda) + 2cov(Z_{20}, \hat{\mathcal{M}}(W_{20}, \lambda))];$$

this implies $\text{err}(\lambda) + 2cov(Z_{20}, \hat{\mathcal{M}}(W_{20}, \lambda))$ is an unbiased estimator of $E[\text{Err}(\lambda)]$ (which is just the expected prediction error); here, the covariance term acts as a bias correction. However, except in simple linear smoothing problems, $cov(Z_{20}, \hat{\mathcal{M}}(W_{20}, \lambda))$ is not easy to calculate or otherwise estimate analytically.

Efron (2004) proposes to use a parametric bootstrap procedure to deal with this problem. Again, consider a fixed λ. Following Efron (2004), suppose we generate the b^{th} bootstrap sample $Z^*_{1i}(b) \sim N(\hat{\mathcal{M}}(W_{1i}, \lambda), \hat{\sigma}^2_{Z_1 - \hat{\mathcal{M}}}), i = 1, \dots, n_1$, where

$$\hat{\sigma}^2_{Z_1 - \hat{\mathcal{M}}} = n_1^{-1} \sum_{i=1}^{n_1} (Z_{1i} - \hat{\mathcal{M}}(W_{1i}, \lambda))^2. \tag{12}$$

For generating boostrap samples, we can use $\hat{m}(\cdot)$ in place of $\hat{\mathcal{M}}(W_{1i}, \lambda)$, as bootstrapping does not depend on λ and this estimation only needs to be done once.

For each $b = 1, \dots, B$ we run our code on $\{(Z^*_{1i}(b), W_{1i}, \hat{Z}_{1i}), i = 1, \dots, n_1\}$, to obtain a new $\hat{\mathcal{M}}^*(w, \lambda)$. We can compute for each $i = 1, \dots, n_1$

$$C^*_i(\lambda) = \frac{1}{B-1} \sum_{b=1}^{B} \hat{\mathcal{M}}^*(W_{1i}, \lambda)(Z^*_{1i}(b) - \bar{Z}^*_{1i}), \tag{13}$$

where

$$\bar{Z}^*_{1i} = \frac{1}{B} \sum_{b=1}^{B} Z^*_{1i}(b) \tag{14}$$

and then define the boostrap corrected error as

$$\text{err}_{cor}(\lambda) = \text{err}(\lambda) + 2 \sum_{i=1}^{n_1} C^*_i(\lambda) \tag{15}$$

If run over a grid of possible λ values, it should be possible to choose the λ that minimizes $err_{cor}(\cdot)$ (or a smoothed version of it).

- Why are we not bootstrapping the entire training set?

Code

Code has been written that implements Strategy 2. For the moment we do not have a test set; thus, the entire dataset is the training set with half for the learning set and half for the evaluation set. To choose λ we have started with the estimate of λ_{opt} in (8) at the root node. We multiply that estimate by a constant c and do a grid search on [0,cλ]. The final $\hat{\lambda}$ is the one gives the best optimism-corrected error rate in (15). Once we have $\hat{\lambda}$ we build a CART tree based on (2).

```
library(randomForest)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(rpart)
library(rpart.plot)
library(logspline)
```

Functions

The goal of this code is to build an interpretable tree by employing the predictive accuracy of a bagged learner.

aim2

The main function is the `aim2` function which has calls:

- **dat** is the data frame to which the model is fit
- **nreps** is set to 1 for the time being
- **ngrid** is the number of lambdas in the grid search
- **mult** is the number multiplied times the intial lambda which gives the maximum lambda in the grid search
- **seed** fixes the random number generator for reproducibility
- **outvar** is the name of the outcome variable in the fitting

The output from the function 'aim2 is:

- **lambdas** are the values of lambda from grid search
- **Error.lambdas** are the bootstrapped corrected errors from (15)
- **error.lambdas** is the apparent prediction error from (11)
- **optimism** is $2 * \sum_i^{n_1} = 1 C_i^*(\lambda)$ from the right hand side of sum in (15)
- **fits** are the new $\hat{\mathcal{M}}(W_{1i}, \lambda)$ for the different lambdas
- **predictions** are the predicted values for the evaluations set from $\hat{\mathcal{M}}(W_{1i}, \lambda)$
- **evaluation.dat** is the evaluation data

```
aim2 <- function(dat,nreps=1,n.grid=20,mult=2,outvar="Y",prop.learning=0.5)
{
  #Functions that go into penalized fitting method
  aim2.list <- list(eval=aim2.eval, split=aim2.split, init=aim2.init,
                    summary=aim2.summary, text=aim2.text)
  n <- nrow(dat)

  #Identify outcome variable   --- this is redundant with functions statement and should be changed
  which.outcome <- which(colnames(dat)==outvar)
  colnames(dat)[which.outcome] <- "outvar.aim2"

  #Split training set into learning and evaluation sets - now based on 50/50 split
```

```
# later look at different alternatives
nlearn <- round(prop.learning*n)
neval <- n-nlearn
samp <- sample(1:n,n,replace=FALSE)
wlearn <- sort(samp[1:nlearn])
weval <- sort(samp[(nlearn+1):n])
learning.dat <- dat[wlearn,]
evaluation.dat <- dat[weval,]

#The lambdas chosen using a grid. Here, get values for variables based on root node
fit.rf.learning <- randomForest(outvar.aim2 ~ .,data = learning.dat) # Fit RF with learning set
predict.rf.evaluation <- predict(fit.rf.learning,newdata=evaluation.dat,
                                 predict.all=TRUE) #  $\widehat{Z}_{1i}$
mean.evaluation <- mean(evaluation.dat$outvar.aim2) # $\mu_{Z_1}$
var.evaluation <- var(evaluation.dat$outvar.aim2) # $\sigma^2_{Z_1}$
zbarhat <- mean(predict.rf.evaluation$aggregate) # $ \bar{\hat{Z_1}}$

  # NOTE: this zbarhat means we are doing the
  # alternative shrinkage target in equation 9 not the original in 6

var.z1s <-  apply(predict.rf.evaluation$individual,1,var) # $\sigma^2_{\hat{Z_1i}}$
alphas <- 1/var.z1s # $\alpha_i$
alphabar <- mean(alphas) # \bar{\alpha}$
lambda <- var.evaluation/(neval*alphabar*(mean.evaluation-zbarhat)^2) #with-in node
                                                                      #choice of \lambda
lambdas <- seq(0,mult*lambda,length.out=n.grid)  # list of possible lambdas
n.lambdas <- length(lambdas) #length of list
error.lambdas <- rep(0,length(lambdas))
fits <- vector("list",n.lambdas)
predictions <- vector("list",n.lambdas)
#print("almost there")
# To get the err(\lambda) - uncorrected - currently equation 11
for(j in 1:n.lambdas)
  {
    #print(lambdas[j])
    current.fit <- rpart(outvar.aim2 ~ .,data = evaluation.dat,
                         parms=list(lambda=lambdas[j],
                         yhat=predict.rf.evaluation$aggregate,
                         alpha=alphas),method=aim2.list)
    xgroup <- rep(1:10, length = nrow(evaluation.dat))
    xfit <- xpred.rpart(current.fit,xgroup)
    xerror <- colMeans((xfit - evaluation.dat$outvar.aim2)^2)
    min.CP<-current.fit$cptable[which(xerror==min(xerror)),1][1]
    current.fit.pruned<-prune(current.fit,cp=min.CP)
    predicted.fit <- predict(object=current.fit.pruned,newdata=evaluation.dat)
    error.lambdas[j] <- sum((evaluation.dat$outvar.aim2-predicted.fit)^2)
    fits[[j]] <- current.fit.pruned
    predictions[[j]] <- predicted.fit
}

# To get the optimism for correcting the err(\lambda)
optimism <- corrected.lambda(dat=evaluation.dat,lambdas=lambdas,
                             list.object=aim2.list,model=fit.rf.learning,
```

```
                                predicted.values=predict.rf.evaluation$aggregate,
                                alphas=alphas,n.boot=10)

  # err(\lambda) corrected
  Error.lambdas <- error.lambdas+optimism

  #return list of interesting variables.
  list(lambdas=lambdas,Error.lambdas=Error.lambdas,error.lambdas=error.lambdas,
       optimism=optimism,fits=fits,predictions=predictions,
       predicted.fit=predicted.fit,evaluation.dat=evaluation.dat)
}
```

At the end of this function, `optimisim` is as written in equation (13) and `Error.lambdas` is in equation (15).

Corrected lambda function

This function is called by `aim2` to get the optimism correction for the prediction error. This implements the parametric boostrap and evaluates equations (12) - (14) and returns the righthand side of (15).

```
corrected.lambda <- function(dat,lambdas,list.object,model,predicted.values,alphas,n.boot=10)
  {
    n1 <- nrow(dat)
    p <- ncol(dat)
    n.lambdas <- length(lambdas)
    cilambda <- matrix(0,n1,n.lambdas)
    boot.dat <- boot.residual <- matrix(NA,n1,n.boot)
    sigmahat <- sqrt(sum((dat$outvar.aim2-predicted.values)^2)/n1) #SD for \hat{\sigma^2_{Z_1-\bigM}}

    for(b in 1:n.boot) boot.dat[,b] <- rnorm(n1,mean=predicted.values,sd=sigmahat)#bootstrap samples
    boot.mean <- matrix(apply(boot.dat,1,mean))  # \bar{Z^*_{1i}}
    for(i in 1:nrow(boot.dat)) boot.residual[i,] <- boot.dat[i,]-boot.mean[i]
    for(b in 1:n.boot)
      {
        new.dat <- dat
        new.dat$outvar.aim2 <- boot.dat[,b]
        for(j in 1:n.lambdas)
          {
            final.fit <- rpart(outvar.aim2 ~ .,data = new.dat,
                               parms=list(lambda=lambdas[j],
                               yhat=predicted.values,alpha=alphas),
                               method=list.object)
            xgroup <- rep(1:10, length = nrow(new.dat))
            xfit <- xpred.rpart(final.fit,xgroup)
            xerror <- colMeans((xfit - new.dat$outvar.aim2)^2)
            min.CP<-final.fit$cptable[which(xerror==min(xerror)),1][1]
            final.fit.pruned<-prune(final.fit,cp=min.CP)

            bigMhat <- predict(object=final.fit.pruned,newdata=new.dat)
            cilambda[,j] <- cilambda[,j]+bigMhat*boot.residual[,b]
          }
      }
    cilambda <- cilambda/(n.boot-1)
    return(2*apply(cilambda,2,sum))
```

```
  }
```

Hand build rpart tree

To begin we call needed libraries and code the rpart functions for init, eval, and split which are specific to our algorithm. To build an rpart tree by hand, a list of functions needs to be fed to the rpart call. That list is referred to as aim2.list, and used with the argument method=aim2.list. Important functions are an initialization function (aim2.init), an evaluation function (aim2.eval), and a splitting function (aim2.split). Note that in aim2.init the y variable contains three columns: the evaluation set outcome variables Z_{1i}, the α's, and the predicted values $\widehat{Z_{1i}}$. In aim2.eval the value of (2) is computed for the chosen split. In aim2.split the optimal split is found. This is done currently by looping through every value of every variable. Future effort will be undertaken to see if the loop can be removed.

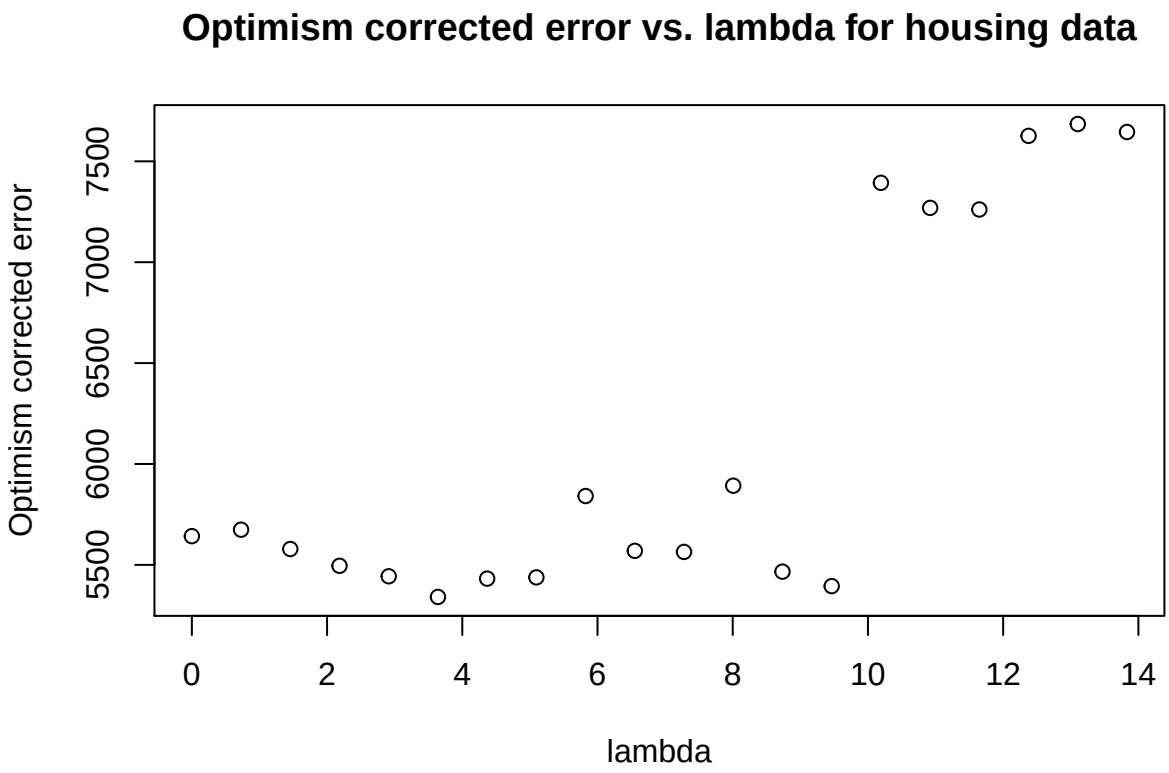
```
#y contains response Z_i, Zhat_i from RF, and alpha where alpha=1/var(zhat)

aim2.init <- function(y, offset, parms, wt)
{
  if (!is.null(offset)) y[,1] <- y[,1]-offset
  list(y=cbind(y,parms$yhat,parms$alpha),parms=parms, numy=3, numresp=1, summary=aim2.summary)
}

aim2.eval <- function(y, wt, parms)
{
  n <- length(y)/3
  lambda <- parms$lambda
  yhat <- y[,2]
  alphas <- y[,3]
  alphabar <- sum(alphas)/n
  y1 <- y[,1]
  r <- 1/(1+lambda*alphabar)
  zbar <- mean(y1)
  zbarhat <- sum(yhat*alphas)/sum(alphas)
  chat <- r*zbar+(1-r)*zbarhat
  rss <- sum((y1-chat)^2+lambda*alphas*(chat-yhat)^2)
  list(label=chat, deviance=rss)
}

aim2.split <- function(y, wt, x, parms, continuous)
  {
    n <- length(y[,1])
    y1 <- y[,1]
    yhat <- y[,2]
    alpha <- y[,3]
    lambda <- parms$lambda
    if (continuous)
      {
        if(is.null(lambda)) compute.lambda #Placeholder until I figure out how to compute lambda
        goodness <- direction <- double(n-1) #Allocate 0 vector
        y.cumsum <- cumsum(y1)
        y.left <- y.cumsum[-n]
        y.right <- y.cumsum[n]-y.left
        yhat.cumsum <- cumsum(yhat*alpha)
        yhat.left <- yhat.cumsum[-n]
```

```
        yhat.right <- yhat.cumsum[n]-yhat.left
        alpha.cumsum <- cumsum(alpha)
        alpha.left <- alpha.cumsum[-n]
        alpha.right <- alpha.cumsum[n]-alpha.left
        for(i in 1:(n-1))
          {
            zbar.left <- y.left[i]/i
            zbar.right <- y.right[i]/(n-i)
            zbarhat.left <- yhat.left[i]/alpha.left[i]
            zbarhat.right <- yhat.right[i]/alpha.right[i]
            alphabar.left <- alpha.left[i]/i
            alphabar.right <- alpha.right[i]/(n-i)
            r.left <- 1/(1+lambda*alphabar.left)
            r.right <- 1/(1+lambda*alphabar.right)
            chat.left <- r.left*zbar.left+(1-r.left)*zbarhat.left
            chat.right <- r.right*zbar.right+(1-r.right)*zbarhat.right

#            goodness[i] <- sum((y1-mean(y1))^2)
#                 - (sum((y1[1:i]-chat.left)^2 +
#                    lambda*alpha[1:i]*(yhat[1:i]-chat.left)^2) +
#                   sum((y1[(i+1):n]-chat.right)^2 +
#                    lambda*alpha[(i+1):n]*(yhat[(i+1):n]-chat.right)^2))
#           Do we need adjustment for missing values like in  vignette example?

            direction[i] <- sign(zbar.left-zbar.right)
            goodness.left <- sum((y1[1:i]-chat.left)^2 + lambda*alpha[1:i]*(yhat[1:i]-chat.left)^2)
            goodness.right <- sum((y1[(i+1):n]-chat.right)^2 +
                                  lambda*alpha[(i+1):n]*(yhat[(i+1):n]-chat.right)^2)
            tss <- sum((y1-mean(y1))^2)
            goodness[i] <- tss-goodness.left-goodness.right

          }
      } # this means we can only have x continuous - no categorical
#    goodness <- 1/goodness
    return(list(goodness=goodness, direction=direction))
  }

aim2.summary <- function(yval, dev, wt, ylevel, digits )
{
  paste(" mean=", format(signif(yval, digits)), ", MSE=" , format(signif(dev/wt, digits)), sep= '')
}

aim2.text <- function(yval, dev, wt, ylevel, digits, n, use.n )
{
  if(use.n) paste(formatg(yval,digits)," nn=", n,sep="")
  else paste(formatg(yval,digits))
}
```

Example: Boston housing data

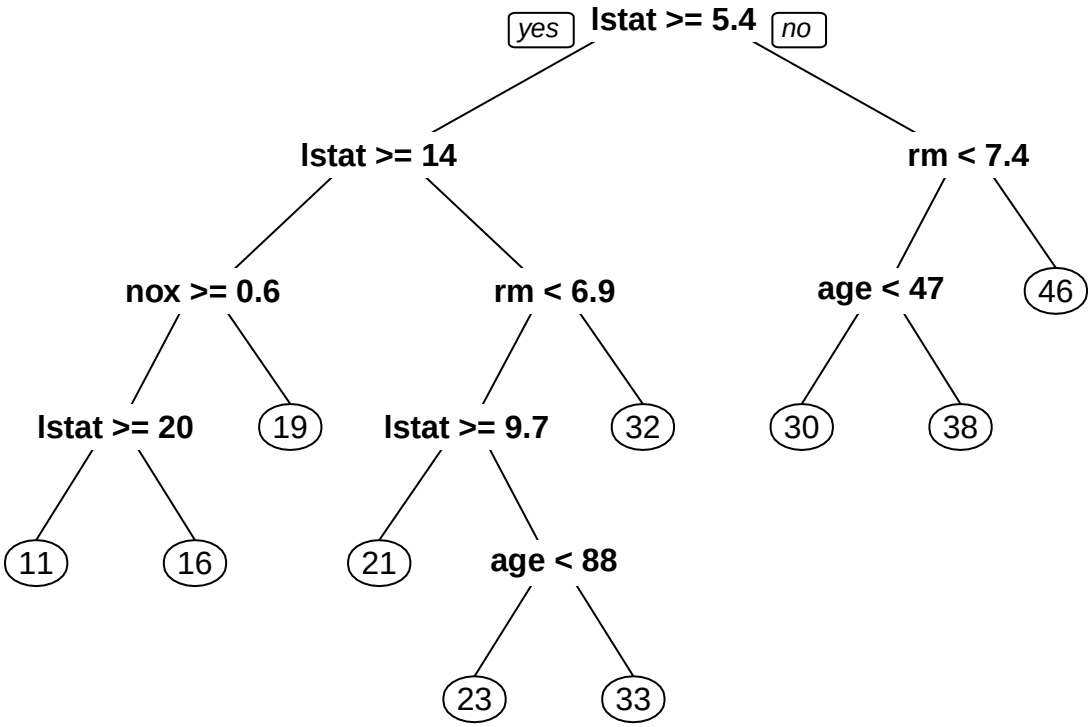
For an example, we have the Boston housing data. In the following block of text we read in the data, rename the columns, run the algorithm (via `aim2` function) and then plot the errors.

```
set.seed(12345)
housing.data <- as.data.frame(matrix(scan("housing.data"),nrow=506,byrow=TRUE))
colnames(housing.data) <- c("crim","zn","indus","chas","nox","rm","age","dis","rad",
                            "tax","ptratiob","b","lstat","mdev")
temp <- aim2(dat=housing.data,nreps=1,n.grid=20,mult=2,outvar="mdev",prop.learning=0.5)

  plot(temp$lambdas,temp$Error.lambdas,xlab="lambda",ylab="Optimism corrected error",
       main="Optimism corrected error vs. lambda for housing data")
```



According to this plot, we would select the lambda equal to 3.6399528 is the one which minimizes the optimism corrected error. Using that lambda we get the following rpart tree:

```
## Warning: Unrecognized rpart object: treating as a numeric response model
```



The resubstitution estimate of prediction error is $ 8798.721$.

If we just ran a rpart tree:

```
summary(justRpart<-rpart(mdev ~ .,data = housing.data)  )
```

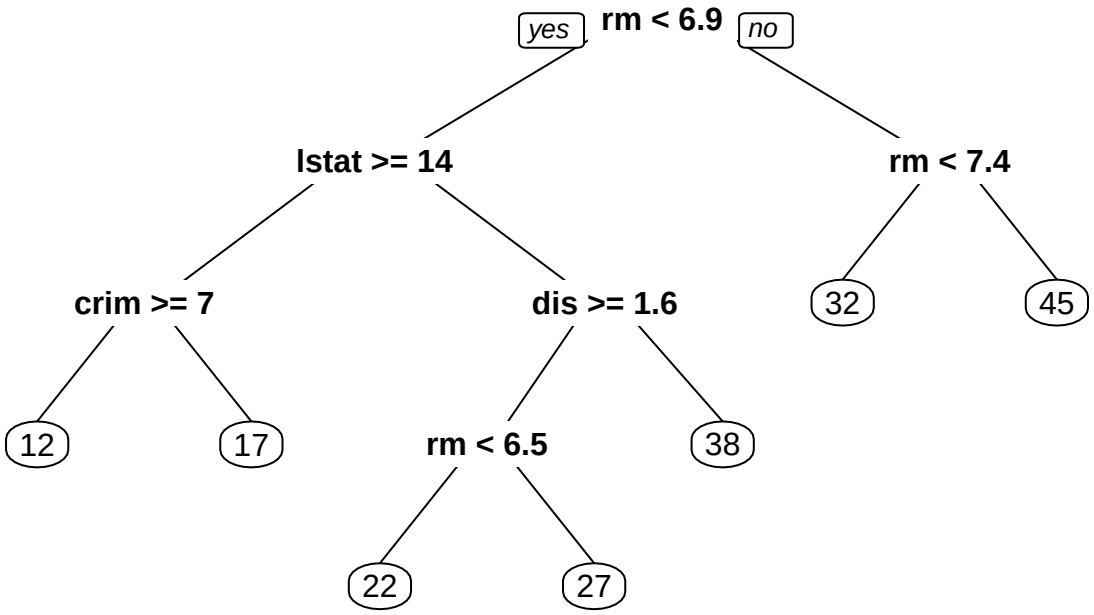
```
## Call:
## rpart(formula = mdev ~ ., data = housing.data)
##   n= 506
##
##           CP nsplit rel error    xerror       xstd
## 1 0.45274420      0 1.0000000 1.0099763 0.08341147
## 2 0.17117244      1 0.5472558 0.6310714 0.05806913
## 3 0.07165784      2 0.3760834 0.4055226 0.04627266
## 4 0.03616428      3 0.3044255 0.3437991 0.04222161
## 5 0.03336923      4 0.2682612 0.3123335 0.04262525
## 6 0.02661300      5 0.2348920 0.3100608 0.04192859
## 7 0.01585116      6 0.2082790 0.2655861 0.03572517
## 8 0.01000000      7 0.1924279 0.2666426 0.03888913
##
## Variable importance
##       rm    lstat      dis    indus      tax ptratiob      nox      age
##       33       21        7        7        6        6        6        6
##     crim       zn      rad        b
##        4        2        1        1
##
## Node number 1: 506 observations,    complexity param=0.4527442
##   mean=22.53281, MSE=84.41956
##   left son=2 (430 obs) right son=3 (76 obs)
##   Primary splits:
##       rm       < 6.941    to the left,  improve=0.4527442, (0 missing)
##       lstat    < 9.725    to the right, improve=0.4423650, (0 missing)
```

```
##       indus    < 6.66     to the right, improve=0.2594613, (0 missing)
##       ptratiob < 19.9     to the right, improve=0.2443727, (0 missing)
##       nox      < 0.6695   to the right, improve=0.2232456, (0 missing)
##   Surrogate splits:
##       lstat    < 4.83     to the right, agree=0.891, adj=0.276, (0 split)
##       ptratiob < 14.55    to the right, agree=0.875, adj=0.171, (0 split)
##       zn       < 87.5     to the left,  agree=0.862, adj=0.079, (0 split)
##       indus    < 1.605    to the right, agree=0.862, adj=0.079, (0 split)
##       crim     < 0.013355 to the right, agree=0.852, adj=0.013, (0 split)
##
## Node number 2: 430 observations,    complexity param=0.1711724
##   mean=19.93372, MSE=40.27284
##   left son=4 (175 obs) right son=5 (255 obs)
##   Primary splits:
##       lstat    < 14.4     to the right, improve=0.4222277, (0 missing)
##       nox      < 0.6695   to the right, improve=0.2775455, (0 missing)
##       crim     < 5.84803  to the right, improve=0.2483622, (0 missing)
##       ptratiob < 19.9     to the right, improve=0.2199328, (0 missing)
##       age      < 75.75    to the right, improve=0.2089435, (0 missing)
##   Surrogate splits:
##       age   < 84.3     to the right, agree=0.814, adj=0.543, (0 split)
##       indus < 16.57    to the right, agree=0.781, adj=0.463, (0 split)
##       nox   < 0.5765   to the right, agree=0.781, adj=0.463, (0 split)
##       dis   < 2.23935  to the left,  agree=0.781, adj=0.463, (0 split)
##       tax   < 434.5    to the right, agree=0.774, adj=0.446, (0 split)
##
## Node number 3: 76 observations,    complexity param=0.07165784
##   mean=37.23816, MSE=79.7292
##   left son=6 (46 obs) right son=7 (30 obs)
##   Primary splits:
##       rm       < 7.437    to the left,  improve=0.5051569, (0 missing)
##       lstat    < 4.68     to the right, improve=0.3318914, (0 missing)
##       ptratiob < 19.7     to the right, improve=0.2498786, (0 missing)
##       rad      < 16       to the right, improve=0.2139402, (0 missing)
##       crim     < 2.742235 to the right, improve=0.2139402, (0 missing)
##   Surrogate splits:
##       lstat    < 3.99     to the right, agree=0.776, adj=0.433, (0 split)
##       ptratiob < 14.75    to the right, agree=0.671, adj=0.167, (0 split)
##       b        < 389.885  to the right, agree=0.658, adj=0.133, (0 split)
##       crim     < 0.11276  to the left,  agree=0.645, adj=0.100, (0 split)
##       indus    < 18.84    to the left,  agree=0.645, adj=0.100, (0 split)
##
## Node number 4: 175 observations,    complexity param=0.026613
##   mean=14.956, MSE=19.27572
##   left son=8 (74 obs) right son=9 (101 obs)
##   Primary splits:
##       crim  < 6.99237  to the right, improve=0.3370069, (0 missing)
##       nox   < 0.607    to the right, improve=0.3307926, (0 missing)
##       dis   < 2.0037   to the left,  improve=0.2927244, (0 missing)
##       tax   < 567.5    to the right, improve=0.2825858, (0 missing)
##       lstat < 19.83    to the right, improve=0.2696497, (0 missing)
##   Surrogate splits:
##       rad      < 16       to the right, agree=0.880, adj=0.716, (0 split)
##       tax      < 567.5    to the right, agree=0.857, adj=0.662, (0 split)
```

```
##       nox      < 0.657    to the right, agree=0.760, adj=0.432, (0 split)
##       dis      < 2.202    to the left,  agree=0.737, adj=0.378, (0 split)
##       ptratiob < 20.15    to the right, agree=0.720, adj=0.338, (0 split)
##
## Node number 5: 255 observations,    complexity param=0.03616428
##   mean=23.3498, MSE=26.0087
##   left son=10 (248 obs) right son=11 (7 obs)
##   Primary splits:
##       dis   < 1.5511   to the right, improve=0.23292420, (0 missing)
##       lstat < 4.91     to the right, improve=0.22084090, (0 missing)
##       rm    < 6.543    to the left,  improve=0.21720990, (0 missing)
##       crim  < 4.866945 to the left,  improve=0.06629933, (0 missing)
##       chas  < 0.5      to the left,  improve=0.06223827, (0 missing)
##   Surrogate splits:
##       crim < 8.053285 to the left,  agree=0.984, adj=0.429, (0 split)
##
## Node number 6: 46 observations,    complexity param=0.01585116
##   mean=32.11304, MSE=41.29592
##   left son=12 (7 obs) right son=13 (39 obs)
##   Primary splits:
##       lstat    < 9.65     to the right, improve=0.3564426, (0 missing)
##       ptratiob < 19.7     to the right, improve=0.2481412, (0 missing)
##       rad      < 7.5      to the right, improve=0.1793089, (0 missing)
##       nox      < 0.639    to the right, improve=0.1663927, (0 missing)
##       indus    < 9.5      to the right, improve=0.1521488, (0 missing)
##   Surrogate splits:
##       crim  < 0.724605 to the right, agree=0.913, adj=0.429, (0 split)
##       nox   < 0.659    to the right, agree=0.913, adj=0.429, (0 split)
##       rad   < 16       to the right, agree=0.891, adj=0.286, (0 split)
##       tax   < 534.5    to the right, agree=0.891, adj=0.286, (0 split)
##       indus < 15.015   to the right, agree=0.870, adj=0.143, (0 split)
##
## Node number 7: 30 observations
##   mean=45.09667, MSE=36.62832
##
## Node number 8: 74 observations
##   mean=11.97838, MSE=14.6744
##
## Node number 9: 101 observations
##   mean=17.13762, MSE=11.39146
##
## Node number 10: 248 observations,    complexity param=0.03336923
##   mean=22.93629, MSE=14.75159
##   left son=20 (193 obs) right son=21 (55 obs)
##   Primary splits:
##       rm       < 6.543    to the left,  improve=0.3896273, (0 missing)
##       lstat    < 7.685    to the right, improve=0.3356012, (0 missing)
##       nox      < 0.5125   to the right, improve=0.1514349, (0 missing)
##       ptratiob < 18.35    to the right, improve=0.1212960, (0 missing)
##       indus    < 4.1      to the right, improve=0.1207036, (0 missing)
##   Surrogate splits:
##       lstat < 5.055    to the right, agree=0.839, adj=0.273, (0 split)
##       crim  < 0.017895 to the right, agree=0.794, adj=0.073, (0 split)
##       zn    < 31.5     to the left,  agree=0.790, adj=0.055, (0 split)
```

```
##       dis   < 10.648   to the left,  agree=0.782, adj=0.018, (0 split)
##
## Node number 11: 7 observations
##   mean=38, MSE=204.1457
##
## Node number 12: 7 observations
##   mean=23.05714, MSE=61.85673
##
## Node number 13: 39 observations
##   mean=33.73846, MSE=20.24391
##
## Node number 20: 193 observations
##   mean=21.65648, MSE=8.23738
##
## Node number 21: 55 observations
##   mean=27.42727, MSE=11.69398
```

```
#prp(justRpart)
min.CP<-justRpart$cptable[which(justRpart$cptable[,4]==min(justRpart$cptable[,4])),1]
justRpartpruned<-prune(justRpart,cp=min.CP)
prp(justRpartpruned)
```




```
  rpart.test.dat.predicted.fit.resub<-predict(object=justRpartpruned,newdata=housing.data)
 rpart.error.test.dat.resub <- sum((housing.data$mdev-rpart.test.dat.predicted.fit.resub)^2)
```

With prediction error 8896.908

Switching to Composite functions.

At the moment we have three composite functions - the original we worked on in DC, the "thirds" version Adam has worked on and Rob's version which searches a grid and uses a logspline density for the bootstrap.

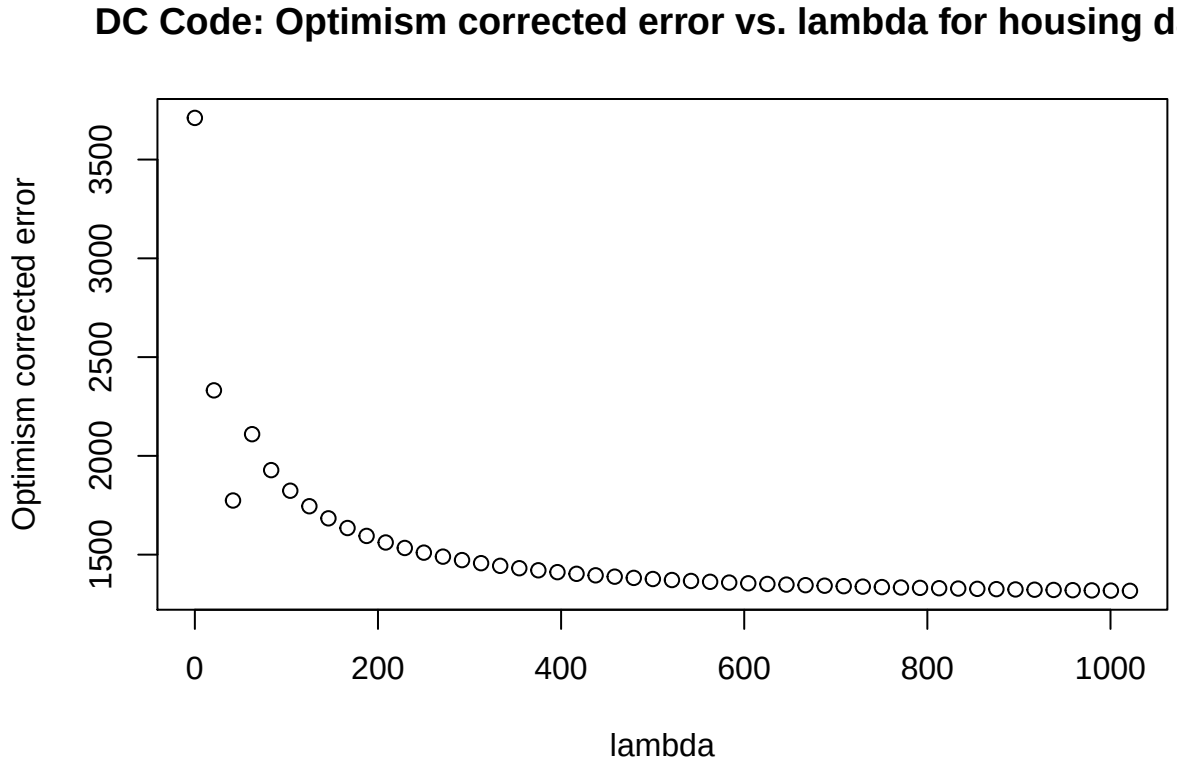
DC Code

```
composite.rpart=function(dat,n.grid=20,mult=2,outvar="Y",prop.learning=0.5)
{
  n <- nrow(dat)
  which.outcome <- which(colnames(dat)==outvar)
  colnames(dat)[which.outcome] <- "outvar.aim2"

  #Split into learning and evaluation sets
  nlearn <- round(prop.learning*n)
  neval <- n-nlearn
  samp <- sample(1:n,n,replace=FALSE)
  wlearn <- sort(samp[1:nlearn])
  weval <- sort(samp[(nlearn+1):n])
  learning.dat <- dat[wlearn,]
  evaluation.dat <- dat[weval,]

  fit.rf.learning <- randomForest(outvar.aim2 ~ .,data = learning.dat) # Fit RF with learning set
  predict.rf.evaluation <- predict(fit.rf.learning,newdata=evaluation.dat,predict.all=TRUE) #  $\wideha
  mean.evaluation <- mean(evaluation.dat$outvar.aim2) # $\mu_{Z_1}$
  var.evaluation <- var(evaluation.dat$outvar.aim2) # $\sigma^2_{Z_1}$
  zbarhat <- mean(predict.rf.evaluation$aggregate) # $ \bar{\hat{Z_1}}$
  var.z1s <-  apply(predict.rf.evaluation$individual,1,var) # $\sigma^2_{\hat{Z_1i}}$
  alphas <- 1/var.z1s # $\alpha_i$
  alphabar <- mean(alphas) # \bar{\alpha}$
  lambda <- var.evaluation/(neval*alphabar*(mean.evaluation-zbarhat)^2) #with-in node
                                                                        #choice of \lambda
  #print(paste("lambda =",lambda))
  lambdas <- seq(0,mult*lambda,length.out=n.grid)  # list of possible lambdas
  n.lambdas <- length(lambdas) #length of list
  error.lambdas <- rep(0,length(lambdas))
  fits <- vector("list",n.lambdas)
  predictions <- vector("list",n.lambdas)
  use.dat<-evaluation.dat
  # To get the err(\lambda) - uncorrected - currently equation 11
  for(j in 1:n.lambdas)
    {
      new.lambda <- lambdas[j]
      #print(new.lambda)
      new.denom <- (1+alphas*new.lambda)
      #print(new.denom)
      ri <- 1/new.denom
      ci <- ri*use.dat$outvar.aim2 + (1-ri)*predict.rf.evaluation$aggregate
      new.use.dat <- use.dat
      new.use.dat$outvar.aim2 <- ci
      current.fit <- rpart(outvar.aim2 ~ .,data = new.use.dat)
      min.CP<-current.fit$cptable[which(current.fit$cptable[,4]==min(current.fit$cptable[,4])),1][1]
      current.fit.pruned<-prune(current.fit,cp=min.CP)
      predicted.fit <- predict(object=current.fit.pruned, data=new.use.dat)
      error.lambdas[j] <- sum((new.use.dat$outvar.aim2-predicted.fit)^2)
      fits[[j]] <- current.fit
      predictions[[j]] <- predicted.fit
    }
```

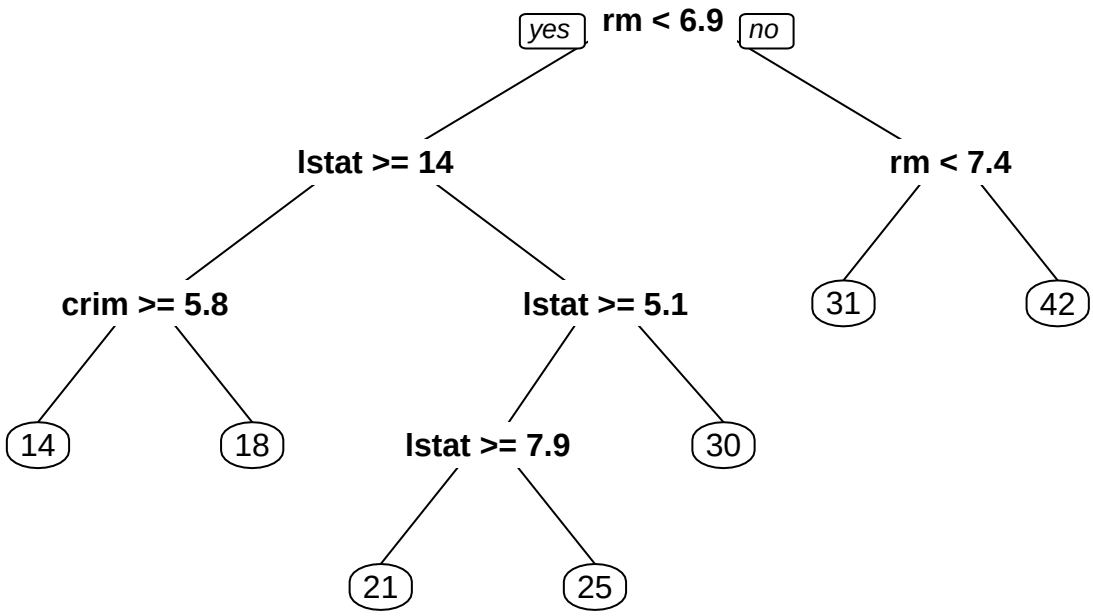
```
  list(lambda=lambda,lambdas=lambdas,error.lambdas=error.lambdas,fits=fits,predictions=predictions)
}


#housing.data <- as.data.frame(matrix(scan("housing.data"),nrow=506,byrow=TRUE))
#colnames(housing.data) <- c("crim","zn","indus","chas","nox","rm","age","dis","rad",
#                            "tax","ptratiob","b","lstat","mdev")
#set.seed(12345)
#temp <- composite.rpart(dat=housing.data,n.grid=20,outvar="mdev")
#plot(temp$lambdas,temp$Error.lambdas,xlab="lambda",ylab="Optimism corrected error",
#     main="Optimism corrected error vs. lambda for housing data")
```

The Boston housing data with the DC Code looks like:

```
temp <- composite.rpart(dat=housing.data,n.grid=50,mult=4,outvar="mdev")
plot(temp$lambdas,temp$error.lambdas,xlab="lambda",ylab="Optimism corrected error",
     main="DC Code: Optimism corrected error vs. lambda for housing data")
```



The min of which is$ 1021.2805946$.



The corresponding tree is:

With corresponding resubstitution estimate of 1.0880917×10^4.

Adam's Code

```
composite.rpart.thirds <- function(dat,n.grid=20,mult=2,outvar="Y")
{
  n <- nrow(dat)
  which.outcome <- which(colnames(dat)==outvar)
  colnames(dat)[which.outcome] <- "outvar.aim2"

  #Split into learning and evaluation sets
  nlearn <- round(n/3)
  ndisc <- round(n/3)
  neval <- n-nlearn-ndisc
  samp <- sample(1:n,n,replace=FALSE)
  wlearn <- sort(samp[1:nlearn])
  wdisc <- sort(samp[(nlearn+1):(nlearn+ndisc)])
  weval <- sort(samp[(nlearn+ndisc+1):n])

  learning.dat <- dat[wlearn,]
  discovery.dat <- dat[wdisc,]
  evaluation.dat <- dat[weval,]

  fit.rf.learning <- randomForest(outvar.aim2 ~ .,data = learning.dat) # Fit RF with learning set
  predict.rf.discovery <- predict(fit.rf.learning,newdata=discovery.dat,predict.all=TRUE) #  $\widehat{
  mean.discovery <- mean(discovery.dat$outvar.aim2) # $\mu_{Z_1}$
  var.discovery <- var(discovery.dat$outvar.aim2) # $\sigma^2_{Z_1}$
  zbarhat <- mean(predict.rf.discovery$aggregate) # $ \bar{\hat{Z_1}}$
  var.z1s <-  apply(predict.rf.discovery$individual,1,var) # $\sigma^2_{\hat{Z_1i}}$
  alphas <- 1/var.z1s # $\alpha_i$
  alphas <- alphas/sum(alphas)
  alphabar <- mean(alphas) # \bar{\alpha}$
  lambda <- var.discovery/(ndisc*alphabar*(mean.discovery-zbarhat)^2) #with-in node
```

```
                                                                    #choice of \lambda
  print(paste("zbarhat =",zbarhat))
  print(paste("mean.discovery =",mean.discovery))
  print(paste("alphabar =",alphabar))
  print(paste("neval =",neval))
  print(paste("var.discovery =",var.discovery))
  print(paste("lambda =",lambda))
  lambdas <- seq(0,mult*lambda,length.out=n.grid)  # list of possible lambdas
  n.lambdas <- length(lambdas) #length of list
  error.lambdas <- rep(0,n.lambdas)
  fits <- vector("list",n.lambdas)
  pruneds <- vector("list",n.lambdas)
  predictions <- vector("list",n.lambdas)
  use.dat<-discovery.dat
  # To get the err(\lambda) - uncorrected - currently equation 11
  for(j in 1:n.lambdas)
    {
      new.lambda <- lambdas[j]
      new.denom <- (1+alphas*new.lambda)
      ri <- 1/new.denom
      ci <- ri*use.dat$outvar.aim2 + (1-ri)*predict.rf.discovery$aggregate
      new.use.dat <- use.dat
      new.use.dat$outvar.aim2 <- ci
      current.fit <- rpart(outvar.aim2 ~ .,data = new.use.dat)
      min.CP<-current.fit$cptable[which(current.fit$cptable[,4]==min(current.fit$cptable[,4])),1][1]
      current.fit.pruned<-prune(current.fit,cp=min.CP)
      predicted.fit <- predict(object=current.fit.pruned, data=evaluation.dat)
      error.lambdas[j] <- sum((evaluation.dat$outvar.aim2-predicted.fit)^2)
      fits[[j]] <- current.fit
      pruneds[[j]] <- current.fit.pruned
      predictions[[j]] <- predicted.fit
    }
  best.lambda <- lambdas[order(error.lambdas)[1]]
  new.denom <- (1+alphas*best.lambda)
  ri <- 1/new.denom
  ci <- ri*dat$outvar.aim2 + (1-ri)*predict(fit.rf.learning,newdata=dat)
  dat$outvar.aim2 <- ci
  current.fit <- rpart(outvar.aim2 ~ .,data = dat)
  min.CP<-current.fit$cptable[which(current.fit$cptable[,4]==min(current.fit$cptable[,4])),1]
  current.fit.pruned<-prune(current.fit,cp=min.CP)

  list(best.lambda=best.lambda,lambda=lambda,lambdas=lambdas,error.lambdas=error.lambdas,fits=fits,prune
}
```

```
temp2 <- composite.rpart.thirds(dat=housing.data,n.grid=20,mult=3,outvar="mdev")
```

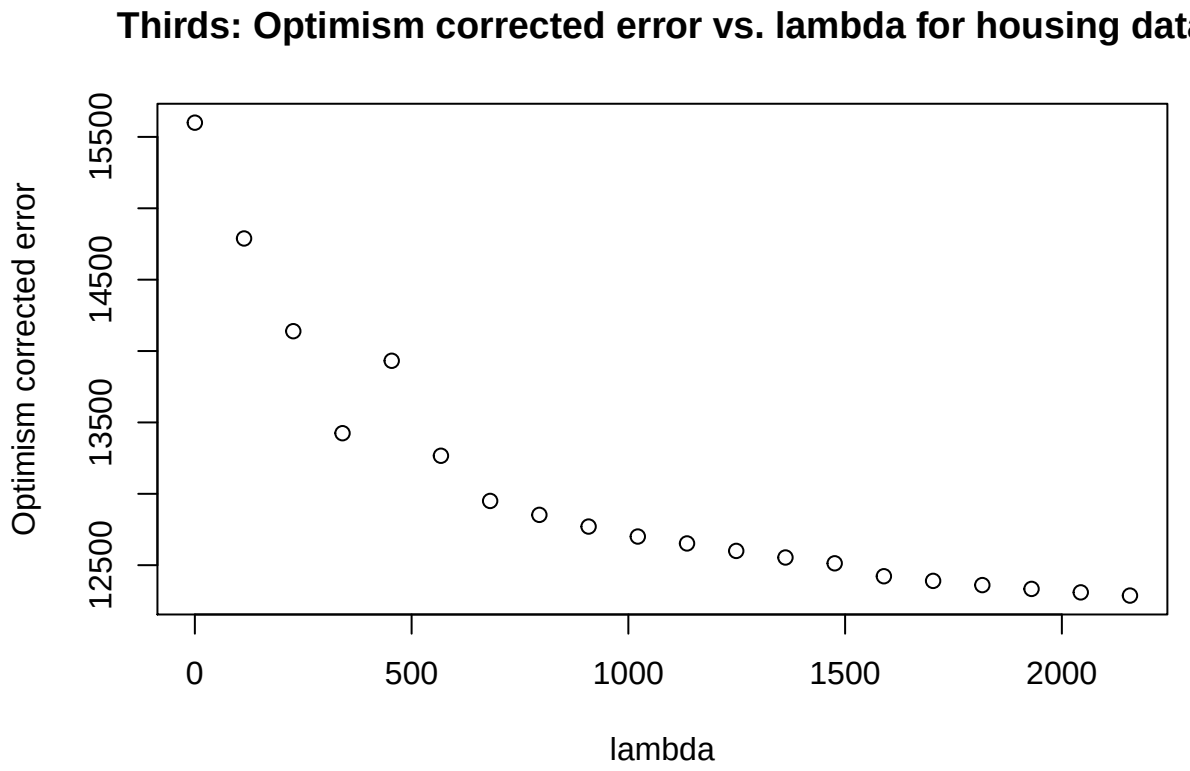
```
## [1] "zbarhat = 22.530026172172"
## [1] "mean.discovery = 22.9023668639053"
## [1] "alphabar = 0.00591715976331361"
## [1] "neval = 168"
## [1] "var.discovery = 99.6863038884193"
## [1] "lambda = 719.042385044249"
```

```
## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
```

```
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
```

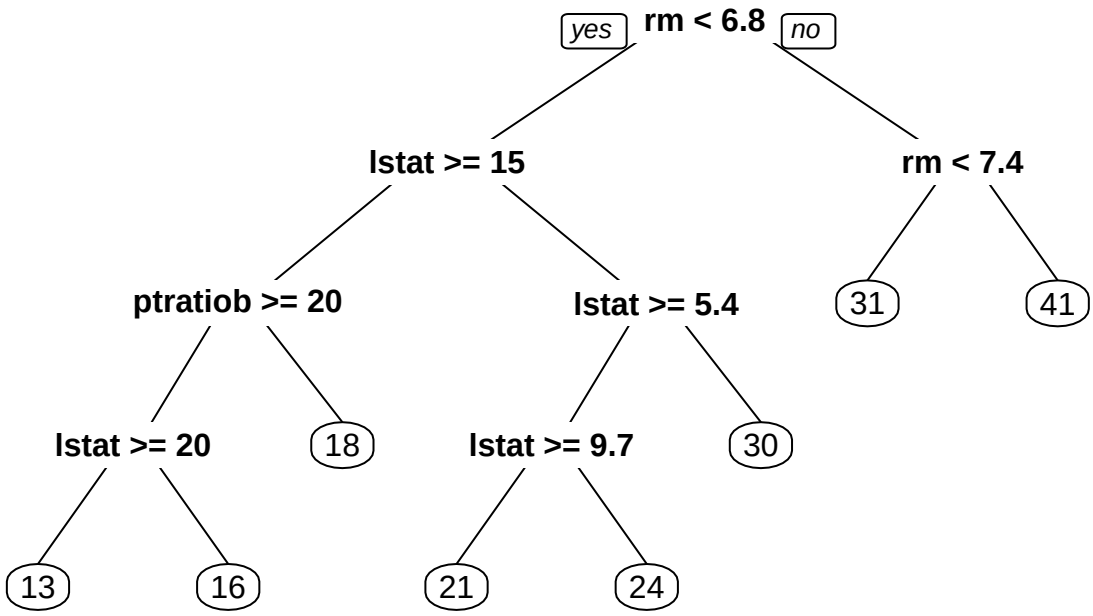
```
## is not a multiple of shorter object length

## Warning in evaluation.dat$outvar.aim2 - predicted.fit: longer object length
## is not a multiple of shorter object length

## Warning in ri * dat$outvar.aim2: longer object length is not a multiple of
## shorter object length

## Warning in (1 - ri) * predict(fit.rf.learning, newdata = dat): longer
## object length is not a multiple of shorter object length
```

```
plot(temp2$lambdas,temp2$error.lambdas,xlab="lambda",ylab="Optimism corrected error",
     main="Thirds: Optimism corrected error vs. lambda for housing data")
```



The min of which is 2157.1271551.

The corresponding tree is:



With corresponding resubstitution estimate of 1.0002355×10^4.

Rob's Code

Comments from Rob:

One modification that I did make is to use a log-spline density estimator in place of a normal distribution for carrying out the parametric bootstrap; you need the package logspline from CRAN. This does quite a good job in capturing and then subsequently simulating data with a response profile that looks similar to the original (definitely better than the normal). But unfortunately the cross-validation curve that gets generated doesn't really differ a lot from that obtained using the normal distribution. Not really sure what is going on here.

The leave one out CV procedure has a clearly defined minimum for lambda for the housing data, at least over the range of lambdas I considered. The parametric bootstrap gives a local min that is in the same ballpark. I haven't looked at the corresponding composite response tree built using lambdas in this range.

If you run it, it will take a while (3-5 mins) since it does both LOO and parametric bootstrap CV (with 1000 replicates – 10 is way too small).

Two other things:

- I don't think the original method for calculating the "root node" lambda is correct for the composite methods so right now we are stuck using a grid.
- The composite response seems to have some potential to create heteroscedasticity – I think low variance predictions will generate composite responses with smaller variances compared to those with high variance predictions, for example. There has been some work on studying the performance of trees in this setting but I don't know what kind of software might be out there to fit such trees.

```
# R version of Matlab function 'repmat'. E.g., if given a
# vector 'X' of length 'k', then repmat(X,1,n) will give the
# k x n matrix where each matrix column is a copy of X.


repmat = function(X, m, n) {
  if (is.vector(X))
```

```
    X = matrix(X, length(X), 1)
  mx = dim(X)[1]
  nx = dim(X)[2]
  return(matrix(t(matrix(X, mx, nx * n)), mx * m, nx * n, byrow = T))
}

# Use leave-one-out CV (LOO)
CVcorrected.lambda <- function(dat, lambdas, model, predicted.values,
                               alphas) {
  n1 <- nrow(dat)
  p <- ncol(dat)
  n.lambdas <- length(lambdas)
  cilambda <- matrix(0, n1, n.lambdas)

  for (b in 1:n1) {
    # create new LOO dataset that deletes bth observation
    new.dat <- dat[-b, ]

    # for bth LOO dataset, loop over lambda
    for (j in 1:n.lambdas) {
      # derive modified response
      new.lambda <- lambdas[j]
      ri <- 1/(1 + alphas[-b] * new.lambda)
      ci <- ri * new.dat$outvar.aim2 + (1 - ri) * predicted.values[-b]

      # create modified LOO dataset
      use.dat = new.dat
      use.dat$outvar.aim2 = ci

      # fit model to modified LOO dataset
      final.fit <- rpart(outvar.aim2 ~ ., data = use.dat)

      ############################################################ prune this fit
      xgroup <- rep(1:10, length = nrow(use.dat))
      xfit <- xpred.rpart(final.fit, xgroup)

      # here -- calculates error using original bootstrap response
      # (new.dat) and not use.dat (weighted response)
      xerror <- colMeans((xfit - new.dat$outvar.aim2)^2)

      ff = which(xerror == min(xerror))[1]
      min.CP <- final.fit$cptable[ff, 1]
      final.fit.pruned <- prune(final.fit, cp = min.CP)
      ############################################################


      # calculate the predicted value for each subject in dat from
      # new LOO model built using dat[-b,]
      bigMhat <- predict(object = final.fit.pruned, newdata = dat)

      # get Zb - tree(-b,Wb) for the bth LOO at lambda = lambda[j]
      # and square the result
      diffpred = (dat$outvar.aim2 - bigMhat)[b]
```

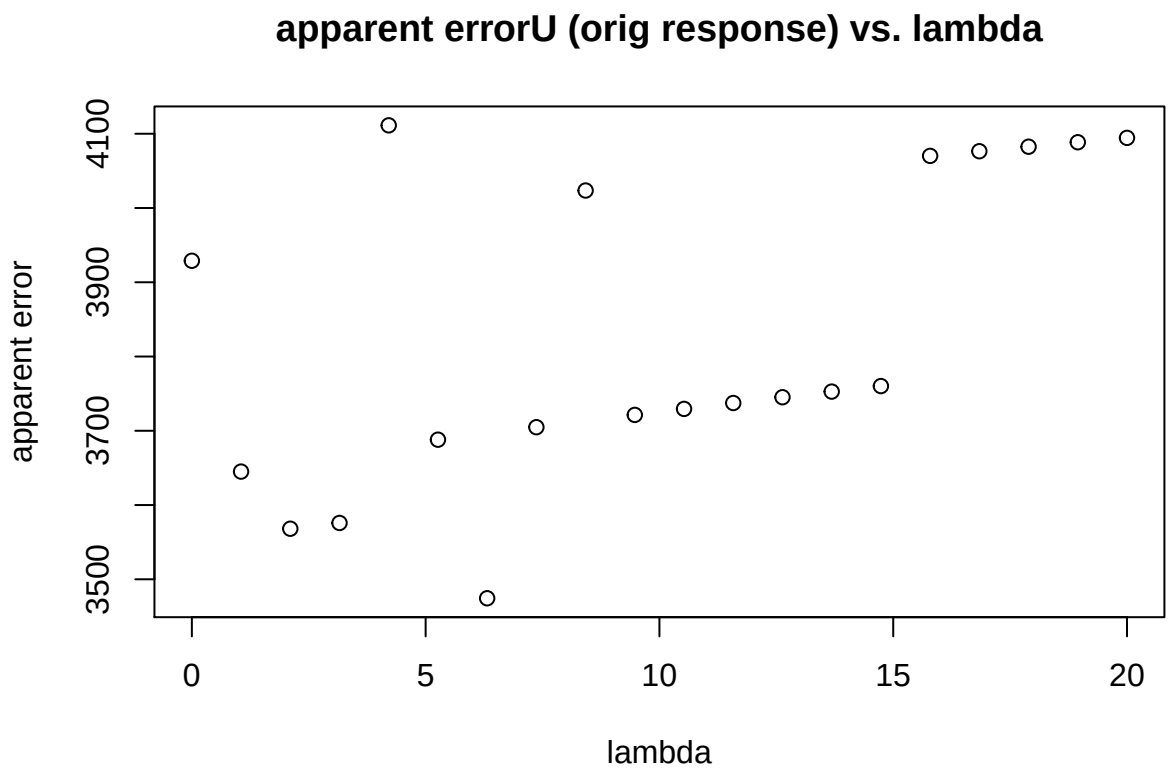
```
      cilambda[b, j] = diffpred^2
    }
  }
  return(apply(cilambda, 2, sum))
}


# parametric bootstrap - this has been modified from Adam's
# original version

corrected.lambda.LSD <- function(dat, lambdas, list.object, model,
                            predicted.values, alphas, n.boot = 10) {
  n1 <- nrow(dat)
  p <- ncol(dat)
  n.lambdas <- length(lambdas)
  cilambda <- matrix(0, n1, n.lambdas)

  # bootstrap the data - generate new 'Z's about \hat{Z}

  # note: the code below is equivalent but faster than original
  # bootstrap code that used normal errors sigmahat <-
  # sqrt(sum((dat$outvar.aim2-predicted.values)^2)/n1) boot.dat
  # =
  # matrix(rnorm(n.boot*n1,mean=predicted.values,sd=sigmahat),n1,n.boot)
  # boot.mean <- apply(boot.dat,1,mean) boot.residual =
  # boot.dat-boot.mean

  # here, use a nonparametric density estimator for the errors
  # 'Zi-\hat{Zi}' and then generate bootstrapped Z by adding
  # random error to \hat{Zi}
  yy = dat$outvar.aim2 - predicted.values
  fit = logspline(yy)
  booterr.dat = matrix(rlogspline(n.boot * n1, fit), n1, n.boot)
  boot.dat = repmat(predicted.values, 1, n.boot) + booterr.dat
  boot.mean <- repmat(as.vector(apply(boot.dat, 1, mean)),
                      1, n.boot)
  boot.residual = boot.dat - boot.mean
  for (b in 1:n.boot) {
    new.dat <- dat
    new.dat$outvar.aim2 <- boot.dat[, b]

    for (j in 1:n.lambdas) {
      new.lambda <- lambdas[j]
      ri <- 1/(1 + alphas * new.lambda)
      ci <- ri * new.dat$outvar.aim2 + (1 - ri) * predicted.values
      use.dat = new.dat
      use.dat$outvar.aim2 = ci

      # fit model to bootstrapped weighted response
      final.fit <- rpart(outvar.aim2 ~ ., data = use.dat)

      ############################################################ prune the bootstrap fit
      xgroup <- rep(1:10, length = nrow(use.dat))
```

```
      xfit <- xpred.rpart(final.fit, xgroup)

      # here -- calculates error using original bootstrap response
      # (new.dat) and not use.dat (weighted response)
      xerror <- colMeans((xfit - new.dat$outvar.aim2)^2)

      ff = which(xerror == min(xerror))[1]
      min.CP <- final.fit$cptable[ff, 1]
      final.fit.pruned <- prune(final.fit, cp = min.CP)
      ############################################################

      # calculate the predicted value for each subject in use.dat;
      # note that it does not matter whether use.dat or new.dat is
      # used here since the response is not used in deriving model
      # prediction
      bigMhat <- predict(object = final.fit.pruned, newdata = use.dat)

      # for lambda=lambda[j], add bootstrap contributions to
      # covariance penalty from bth sample. The boot.residual is
      # the bootstrapped response, not weighted response. Also,
      # note that this is numerically equivalent to calculation
      # that replaces 'bigMhat' with 'bigMhat-predicted.values'

      cilambda[, j] <- cilambda[, j] + bigMhat * boot.residual[,
                                                               b]

    }
  }
  cilambda <- cilambda/(n.boot - 1)
  return(2 * apply(cilambda, 2, sum))
}

composite.rpart.Grid = function(dat, n.grid = 20, mult = 1, uplim = 10,
                           outvar = "Y", prop.learning = 0.5) {

  n <- nrow(dat)
  which.outcome <- which(colnames(dat) == outvar)
  colnames(dat)[which.outcome] <- "outvar.aim2"

  # Split into learning and evaluation sets
  nlearn <- round(prop.learning * n)
  neval <- n - nlearn
  samp <- sample(1:n, n, replace = FALSE)
  wlearn <- sort(samp[1:nlearn])
  weval <- sort(samp[(nlearn + 1):n])
  learning.dat <- dat[wlearn, ]
  evaluation.dat <- dat[weval, ]

  # use learning set to derive RF predictions
  fit.rf.learning <- randomForest(outvar.aim2 ~ ., data = learning.dat,
                                  ntree = 1000)
  predict.rf.evaluation <- predict(fit.rf.learning, newdata = evaluation.dat,
                                   predict.all = TRUE)
```

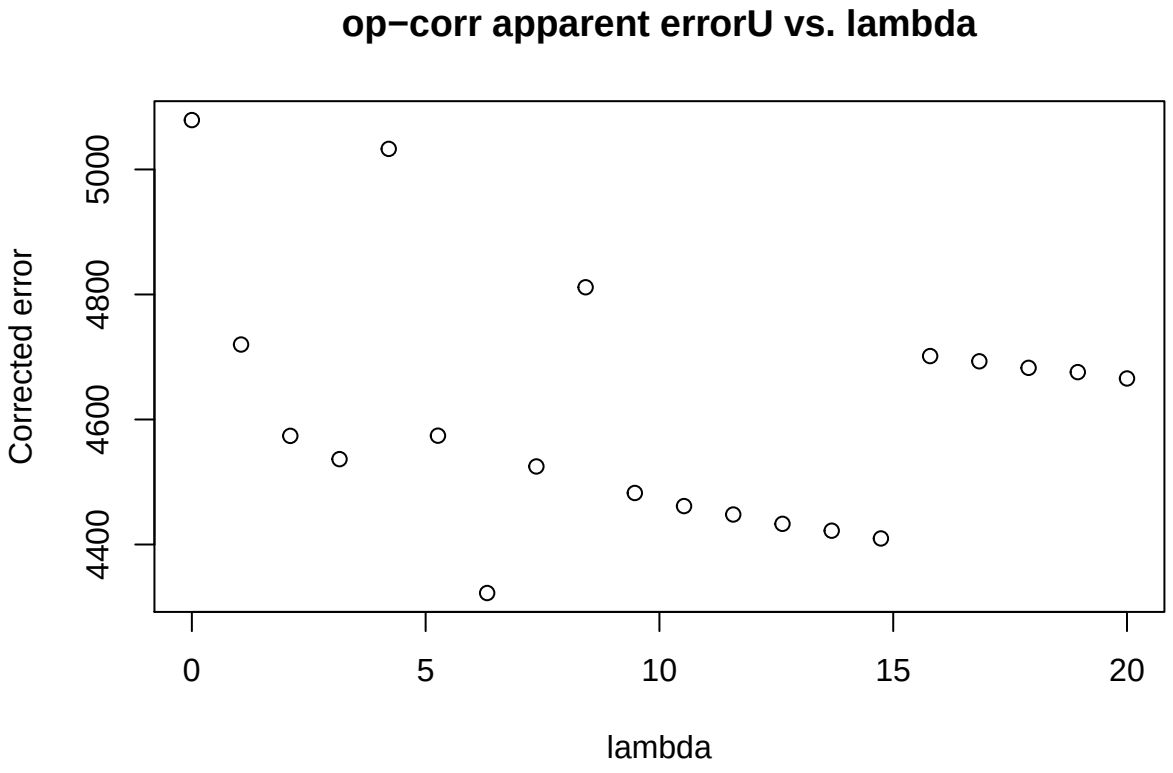
```r
# mean, variance of Z's in evaluation set
mean.evaluation <- mean(evaluation.dat$outvar.aim2)  # $\mu_{Z_1}$
var.evaluation <- var(evaluation.dat$outvar.aim2)  # $\sigma^2_{Z_1}$

# mean, variance of RF predictions for evaluation set
zbarhat <- mean(predict.rf.evaluation$aggregate)  # $ \bar{\hat{Z_1}}$
var.z1s <- apply(predict.rf.evaluation$individual, 1, var)  # $\sigma^2_{\hat{Z_1i}}$

# Hard to know how to chooose. Need to think about objective.
# But sensible choice in that high variance predictions would
# get lower weight
alphas <- 1/var.z1s
# alphas = rep(1,length(var.z1s))

# 'optimal' 'root node' lambda for specified alphas - not
# really relevant here since calculated using a different
# objective. Could revisit this calculation. alphabar <-
# mean(alphas) # \bar{\alpha}$ lambda <-
# var.evaluation/(neval*alphabar*(mean.evaluation-zbarhat)^2)
# print(paste('lambda =',lambda))

# grid of lambda values
lambdas <- seq(0, uplim, length.out = n.grid)  # list of possible lambdas
n.lambdas <- length(lambdas)  #length of list

error.lambdas <- rep(0, length(lambdas))
errorU.lambdas <- rep(0, length(lambdas))
fits <- vector("list", n.lambdas)
predictions <- vector("list", n.lambdas)

use.dat <- evaluation.dat

for (j in 1:n.lambdas) {

  new.lambda <- lambdas[j]
  new.denom <- (1 + alphas * new.lambda)
  ri <- 1/new.denom

  # weighted response calculation
  ci <- ri * use.dat$outvar.aim2 + (1 - ri) * predict.rf.evaluation$aggregate

  # build tree using dataset that employs weighted response
  new.use.dat <- use.dat
  new.use.dat$outvar.aim2 <- ci
  current.fit <- rpart(outvar.aim2 ~ ., data = new.use.dat)

  # prune that tree using manual CV
  min.CP <- current.fit$cptable[which(current.fit$cptable[,
                                                          4] == min(current.fit$cptable[, 4])), 1][1]
  current.fit.pruned <- prune(current.fit, cp = min.CP)

  # derive predicted outcomes for each observation in that
  # dataset
```

```
    predicted.fit <- predict(object = current.fit.pruned,
                             data = new.use.dat)

    # calculate 'apparent error' using 'ci' as response
    error.lambdas[j] <- sum((new.use.dat$outvar.aim2 - predicted.fit)^2)

    # calculate 'apparent error' using actual response
    errorU.lambdas[j] <- sum((use.dat$outvar.aim2 - predicted.fit)^2)


    fits[[j]] <- current.fit
    predictions[[j]] <- predicted.fit
  }

  CVError.lambdas <- CVcorrected.lambda(dat = evaluation.dat,
                                        lambdas = lambdas, model = fit.rf.learning, predicted.values = p
                                        alphas = alphas)

  Error.lambdas <- errorU.lambdas
  optimism <- corrected.lambda.LSD(dat = evaluation.dat, lambdas = lambdas,
                                   list.object = aim2.list, model = fit.rf.learning, predicted.values = pre
                                   alphas = alphas, n.boot = 1000)
  Error.lambdas <- Error.lambdas + optimism

  list(lambdas = lambdas, error.lambdas = error.lambdas, errorU.lambdas = errorU.lambdas,
       Error.lambdas = Error.lambdas, fits = fits, predictions = predictions,
       optimism = optimism, CVError.lambdas = CVError.lambdas)
}
```

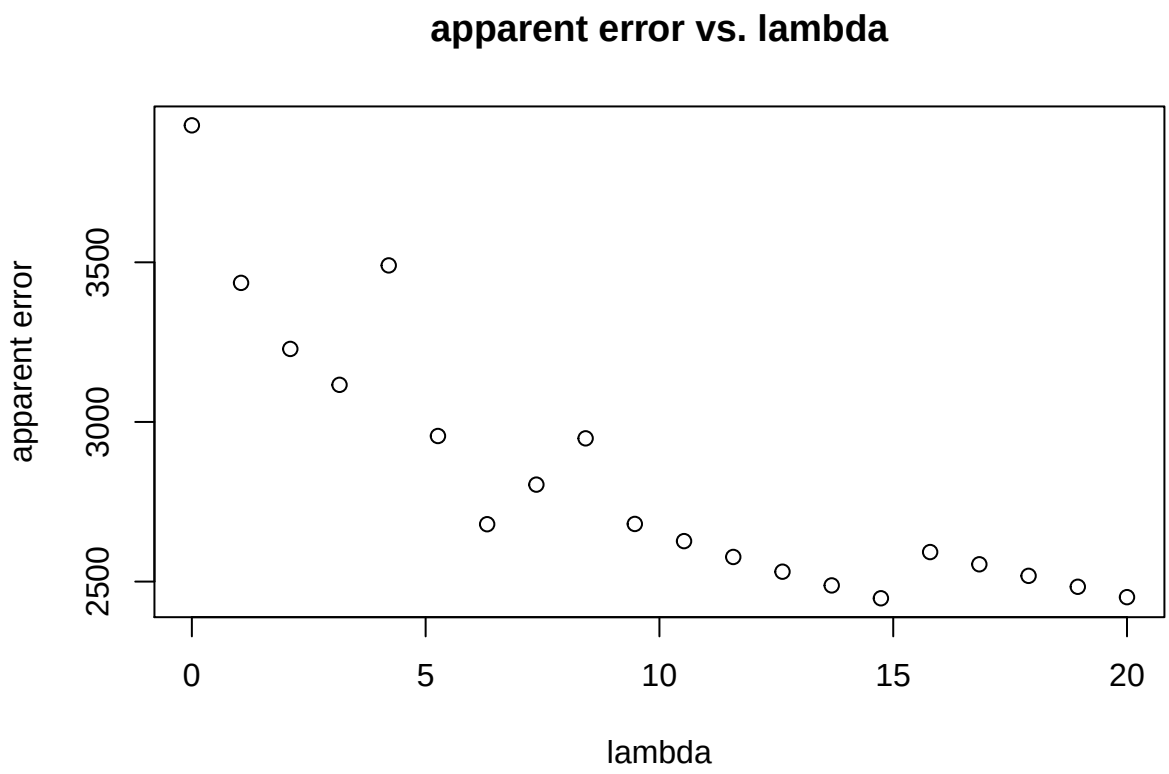
```
temp3 <- composite.rpart.Grid(dat = housing.data, n.grid = 20, mult = 1,
                              uplim = 20, outvar = "mdev", prop.learning = 0.5)


plot(temp3$lambdas, temp3$errorU.lambdas, xlab = "lambda",
     ylab = "apparent error", main = "apparent errorU (orig response) vs. lambda")
```



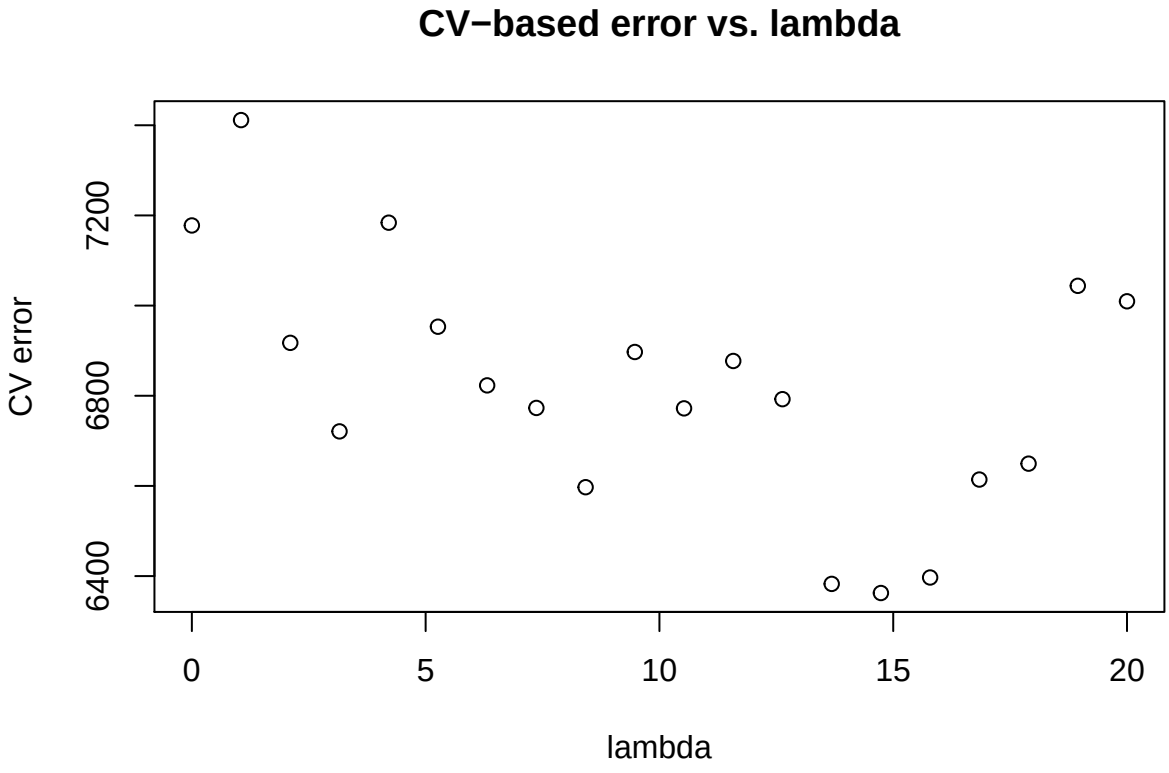

```
plot(temp3$lambdas, temp3$Error.lambdas, xlab = "lambda", ylab = "Corrected error",
     main = "op-corr apparent errorU vs. lambda")
```

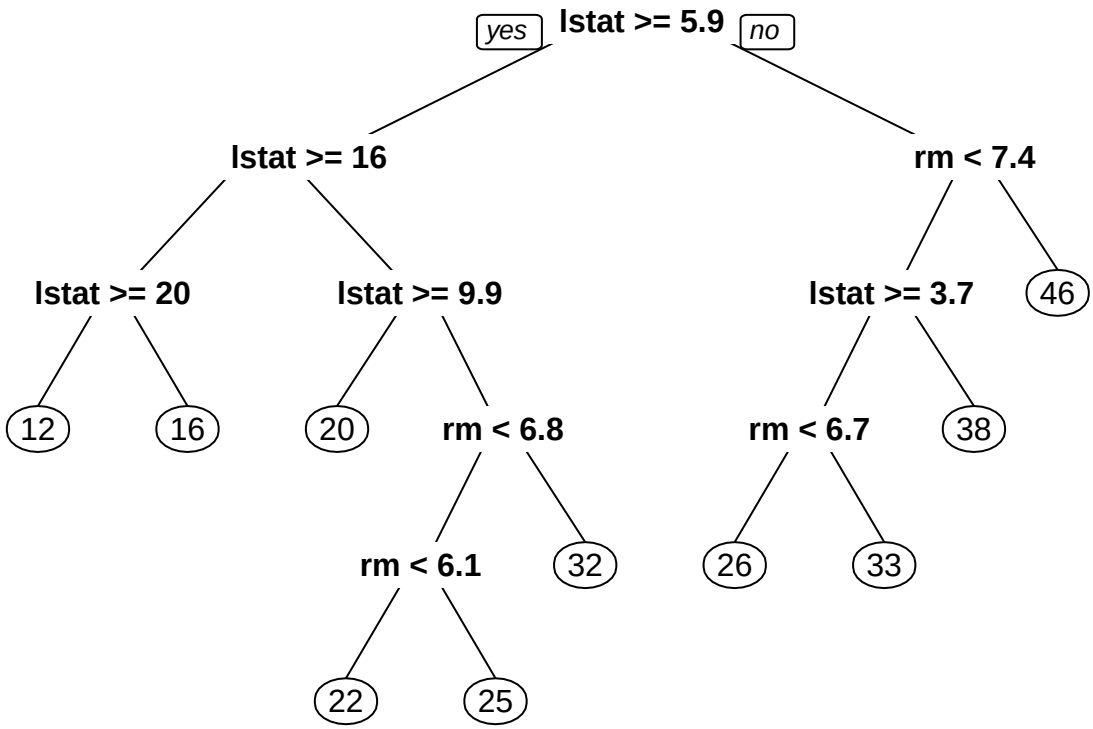



```
plot(temp3$lambdas, temp3$error.lambdas, xlab = "lambda", ylab = "apparent error",
     main = "apparent error vs. lambda")
```




```
plot(temp3$lambdas, temp3$CVError.lambdas, xlab = "lambda",
     ylab = "CV error", main = "CV-based error vs. lambda")
```





The corresponding tree is:

With corresponding resubstitution estimate of 9476.6144304.

Appendix: Important background calculations

Make sure learning/test is updated and that $\mathcal{M}$ is used for final model

Background for estimating c_L and c_R

Let $\omega_i, i \geq 1$ be nonnegative weights, where at least one is positive. Let $\alpha_i, i \geq 1$ and $A_i, i \geq 1$ respectively be sequences of positive and real-valued constants. Let $Z_i, i \geq 1$ be a sequence of random variables. Finally let $\lambda > 0$ be given and consider the problem of minimizing

$$Q(c) = \sum_i \omega_i \left[(Z_i - c)^2 + \lambda \alpha_i (c - A_i)^2 \right]$$

in c. If we differentiate $Q(c)$ with respect to c, set $Q'(c) = 0$, and solve for c, we obtain

$$c(\lambda) = \frac{\sum_i \omega_i Z_i}{\lambda \sum_i \omega_i \alpha_i + \sum_i \omega_i} + \frac{\lambda \sum_i \omega_i \alpha_i A_i}{\lambda \sum_i \omega_i \alpha_i + \sum_i \omega_i}.$$

Doing a bit of algebra,

$$c(\lambda) = r(\lambda) \frac{\sum_i \omega_i Z_i}{\sum_i \omega_i} + (1 - r(\lambda)) \frac{\sum_i \omega_i \alpha_i A_i}{\sum_i \omega_i \alpha_i}. \tag{16}$$

where

$$r(\lambda) = \frac{\sum_i \omega_i}{\lambda \sum_i \omega_i \alpha_i + \sum_i \omega_i} = \frac{1}{(1 + \lambda \bar{\alpha})}, \tag{17}$$

and

$$\bar{\alpha} = \frac{\sum_i \omega_i \alpha_i}{\sum_i \omega_i}. \tag{18}$$

Clearly, $r(\lambda) \in [0, 1]$ and so this is just a shrinkage estimator that balances the observed weighted average (which we'd get if $\lambda = 0$)

$$\frac{\sum_i \omega_i Z_i}{\sum_i \omega_i}$$

with the $\alpha-$modified weighted average of the As

$$\frac{\sum_i \omega_i \alpha_i A_i}{\sum_i \omega_i \alpha_i}.$$

(which we'd get as $\lambda \to \infty$).

Derivation of Within-Node Prediction Error

Now, let $\tilde{Z}$ be independent of $H = \{Z_i, i \geq 1\}$. We can define the conditional prediction error using $c(\lambda)$ as

$$CPE(\lambda) = E\left[(\tilde{Z} - c(\lambda))^2 | H\right]$$

and the prediction error $PE(\lambda) = E_H\left[CPE(\lambda)\right]$ (here, E_H denotes the expectation wrt distribution of H). We would like to know what λ minimizes $PE(\lambda)$. Note that $c(\lambda)$ is known given H under the assumptions made at the beginning of the previous subsection.

We can write

$$(\tilde{Z} - c(\lambda))^2 = \tilde{Z}^2 - 2\tilde{Z}c(\lambda) + [c(\lambda)]^2.$$

Defining $\sigma_Z^2 = var(\tilde{Z})$ and $\mu_Z = E[\tilde{Z}]$ we have

$$CPE(\lambda) = \sigma_Z^2 + \mu_Z^2 - 2\mu_Z c(\lambda) + [c(\lambda)]^2.$$

Hence

$$PE(\lambda) = \sigma_Z^2 + \mu_Z^2 - 2\mu_Z E_H[c(\lambda)] + E_H[[c(\lambda)]^2].$$

Let $\mu_c(\lambda) = E_H[c(\lambda)]$ and $\sigma_c^2(\lambda) = var_H(c(\lambda))$; then, we can rewrite this last expression as

$$PE(\lambda) = \sigma_Z^2 + \mu_Z^2 - 2\mu_Z \mu_c(\lambda) + \sigma_c^2(\lambda) + \mu_c^2(\lambda).$$

Now, suppose $E[Z_i] = \delta$ for each i; then,

$$\mu_c(\lambda) = r(\lambda)\delta + K_1(1 - r(\lambda)).$$

for

$$K_1 = \frac{\sum_i \omega_i \alpha_i A_i}{\sum_i \omega_i \alpha_i}.$$

Similarly, if $var(Z_i) = \gamma$ for each i, then

$$var_c(\lambda) = r^2(\lambda) K_2 \gamma$$

for

$$K_2 = \frac{\sum_i \omega_i^2}{[\sum_i \omega_i]^2}$$

As result we may write

$$PE(\lambda) = \sigma_Z^2 + \mu_Z^2 - 2\mu_Z[r(\lambda)\delta + K_1(1 - r(\lambda))] + r^2(\lambda)K_2\gamma + [r(\lambda)\delta + K_1(1 - r(\lambda))]^2$$

In the special case where $\delta = \mu_Z$ and $\gamma = \sigma_Z^2$, differentiating $PE(\lambda) = 0$ with respect to λ and solving $PE'(\lambda) = 0$ gives

$$\lambda_0 = \frac{K_2 \sigma_Z^2}{\bar{\alpha}(\mu_z - K_1)^2}, \tag{19}$$

where $\bar{\alpha}$ is given in (18).

To connect the notation of the Aim2b set-up and the notation of Section , let $\omega_i = I\{W_{1i} \in Q_v(j,s)\}$, $Z_i = Z_{1i}$ and $A_i = \hat{Z}_{1i}$.

Alternative view of Strategy 2

.

Let $\omega_i = I\{W_{1i} \in Q_v(j,s)\}$, $Z_i = Z_{1i}$ and $A_i = A_{v,j,s} I\{W_{1i} \in Q_v(j,s)\}$ (i.e., $A_{v,j,s}$ doesn't depend on i and thus is constant within node). Then, the formulas (16)-(18) of Section give

$$\widehat{c}_{v,j,s}(\lambda) = r_{v,j,s}(\lambda)\bar{Z}_v(j,s) + (1 - r_{v,j,s}(\lambda))\hat{\bar{A}}_{1v}(j,s),$$

where $r_{v,j,s}(\lambda) = 1/(1 + \lambda\bar{\alpha}_{v,j,s})$,

$$\hat{\bar{A}}_{1v}(j,s) = \{\sum_i I(W_{1i} \in Q_v(j,s))\alpha_i A_{v,j,s}\}/\{\sum_i I(W_{1i} \in Q_v(j,s))\alpha_i\} = A_{v,j,s}$$

and

$$\bar{\alpha}_{v,j,s} = n_{v,j,s}^{-1} \sum_i I(W_{1i} \in Q_v(j,s))\alpha_i$$

where $n_{v,j,s} = \sum_i I(W_{1i} \in Q_v(j,s))$.

```
Applying the results of Section \ref{PE} and assuming all c
calculations are conditional on $W_{1i}, i \geq 1$ it can b
\[
K_2 = n^{-1}_{v,j,s}
\]
and
\[
K_1 = A_{v,j,s}.
\]
Assuming that $E(Z_i) = \mu_Z$ and $var(Z_i) = \sigma^2_Z$
$I(W_{1i} \in Q_{v}(j,s) ) =1$ (i.e., constant mean and va
node), the ``best'' within-node choice of $\lambda$ via
\eqref{lam-opt2} becomes
\[
\lambda_{opt} = \frac{n^{-1}_{v,j,s} \sigma^2_Z}{ \bar{\al
\]

Notice that selecting
\[
A_{v,j,s} = \hat{\bar{Z}}_{1v}(j,s) =
\{ \sum_i I(W_{1i} \in Q_{v}(j,s)) \alpha_i  \hat{Z}_{1i} \
/ \{ \sum_i I(W_{1i} \in Q_{v}(j,s)) \alpha_i \}
\]
gives the same results as in the last section. Selecting i
\[
```

```
A_{v,j,s} = \hat{\bar{Z}}_{1v}(j,s) =
\{ \sum_i I(W_{1i} \in Q_{v}(j,s))  \hat{Z}_{1i} \}
/ \{ \sum_i I(W_{1i} \in Q_{v}(j,s))  \}
\]
gives an alternative shrinkage target.  There are other ch

The point here is that Strategy 2 can be viewed as a proce
shrinking the node-specific estimates towards some node-sp
average predicted value.


#To-Do List

For the short term
1. Decide if want to use alternate shrinkage target or ori
2. Update code so last column of data is not assumed to be
3. Currently - can only have x as continuous - in aim2.spli
```