# Identifying myocardial infarction risk factors in the Wisconsin Longitudinal Survey to aid in intervention program design

*Fred Boehm, Statistics 998*

*March 31, 2015*

## Todo list



### Abstract

### Introduction

Coronary artery disease (CAD) is a leading cause of death in the United States and much of North America and Europe. In 2011, one American died of CAD every 40 seconds, on average, and 155,000 of those deaths were people aged less than 65 years.[1] One manifestation of CAD is a myocardial infarction (MI), which is also called a "heart attack". A MI results from a clot in a coronary artery that diminishes blood flow to the heart muscle, or myocardium. If blood flow disruption persists for a sufficiently long time, the muscle may die, or infarct. The irreparable dead heart muscle diminishes the overall ability of the heart to pump blood. Severe MIs may lead to a patient's death.

Epidemiologists have identified modifiable and non-modifiable risk factors that contribute to CAD risk. Smoking is among the strongest modifiable risk factors, and is thought to elevate CAD risk by triggering elevations in inflammatory molecules in the bloodstream. _____

What is mechanism for smoking causing CAD?

. Diabetes mellitus and hypertension (systolic or diastolic) are typically considered non-modifiable risk factors, although their contribution to CAD risk may be reduced in patients who undertake dramatic lifestyle interventions, such as exercise programs and diet with weight loss. Non-modifiable risk factors include age, a family history of CAD and presence of certain genetic variants

what are other known risk factors?

[1] Mozaffarian et al., "Executive Summary."

.

Our collaborators at the Wisconsin Longitudinal Study (WLS) have undertaken an investigation on a subset of WLS participants with the goal of identifying CAD risk factors in the WLS study population. The ultimate goal of this project is to develop an intervention program to reduce CAD morbidity and mortality in Wisconsin. The investigators would like to extend such an intervention program to Wisconsin residents who are not WLS subjects. Our goal in this report is to identify risk factors for MI among WLS participants.

## Study design

The Wisconsin Longitudinal Study (WLS) is a long-term study of a random sample of 10,317 men and women who graduated from Wisconsin high schools in 1957. According to the WLS website "WLS provides an opportunity to study the life course, intergenerational transfers and relationships, family functioning, physical and mental health and well-being, and morbidity and mortality from late adolescence through 2011."[2]

need more info on WLS?

Our collaborators collected data from the original respondents or their parents in 1957, 1964, 1975, 1992, 2004, and 2011; from a selected sibling in 1977, 1994, 2005, and 2011; from the spouse of the original respondent in 2004; from the spouse of the selected sibling in 2006; and from widow(er)s of the graduates and siblings in 2006.

While the WLS's original purpose didn't focus on cardiovascular health, we feel that the richness of the survey data provide a unique opportunity to identify CAD risk factors for the purpose of informing intervention program design.

## Data description

Our collaborators shared with us a data set that contains records for 19095 individuals (including original subjects and siblings) with 310 variables per subject. 9363 subjects responded (with yes or no) to the 2011 question of whether they had ever had a heart attack.

need to elaborate on data description here.

## Exploratory data analyses

Since our WLS data included 310 variables, we won't provide summaries for all of them in this document. Instead, we focus our exploratory analyses on our response variables (HAC2011, HAC2004, HACinc, doc2011, doc2004, docinc) and covariates that other researchers have identified as associated with coronary artery disease.

Age is known, from epidemiologic studies, to be a strong risk factor for CAD, with older individuals having an elevated CAD risk. We thus examined correlations between the outcome variables and age.

what to explore for categorical data? Only the proportion in each category????

[2] "Wisconsin Longitudinal Study".

| Framingham study variable | WLS Variable |
|---|---|
| Sex | Sex |
| Quantitative total cholesterol | highchol2011, highchol2004 |
| Quantitative HDL cholesterol | None |
| Smoking | smokever2011 (Columns 61 to 87 aim to quantify smoking) |
| Diabetes | diabetes2004, diabetes2011 |
| Age | Age |
| Systolic BP | highbp2004, highbp2011* |
| Treated for high blood pressure | None |

Table 1: Framingham study variables and their closest analogs in WLS. (* SBP not available, so we used reported "high BP".)

## Statistical modeling

We used statistical modeling to try to identify covariates that associated with six distinct outcomes: 1) HAC2004, 2) HAC2011, 3) DOC2004, 4) DOC2011, 5) new self-reported heart attacks (from 2004 to 2011) and 6) new heart attack per doctor's report (from 2004 to 2011). For a subject to qualify as a "new" self-reported heart attack, they must have responded "No" in 2004 and "Yes" in 2011. Analogous definition applies for "new" doctor-reported heart attack. We also noted that some subjects responded in an inconsistent manner, with HAC2004 being yes, but HAC2011 being no. Despite this, we treated the data as though there were no mistaken responses and didn't exclude subjects based on apparent inconsistencies.

We found that HAC2004 had 11534 non-missing responders (with 665 responding "Yes" and 10869 responding "No"). Counts for other variables are provided in Appendix A.

### Statistical modeling with Framingham study predictors

We identified those variables in the WLS that closely match those in the Framingham study[3] ( Table 1). It's important to note that the Framingham study used survival analysis methods, including Cox proportional hazards regression, to identify risk factors for a cardiovascular event. Thus, their study design, analysis, and purpose differ from ours.

Because the risk factors from the Framingham study have strong effect sizes, are easily interpreted, and widely used by both physicians and public health scientists, we decided to perform logistic regression analyses with solely those WLS variables that most closely matched the Framingham variables. We analyzed each of the 6 outcomes of interest. For each outcome variable, we fitted a logistic regression model using the entire data set (omitting those subjects with missing data). We then transformed the fitted logit values to probabilities before plotting a receiver operating characteristic (ROC) curve for each model, in which we use the fitted probabilities to examine the trade-off between specificity and sensitivity. We also calculated the area under the curve (AUC) for each ROC curve.[4]

### Subgroup-specific modeling

## Results

Show a single ROC plot & explain what is meant by "AUC"

[3] D'Agostino et al., "General Cardiovascular Risk Profile for Use in Primary Care the Framingham Heart Study."
[4] Robin et al., "PROC."

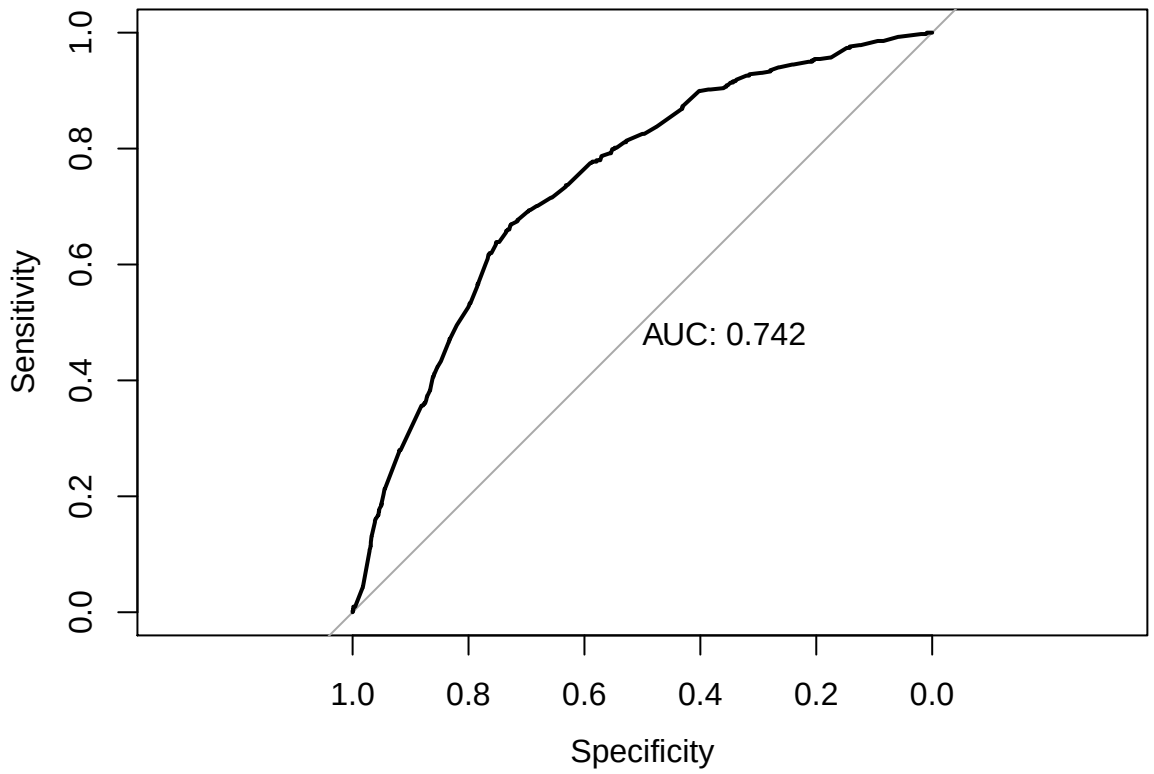


Figure 1: Receiver operating characteristic curve for the logistic regression model with outcome HAC2011 and the six Framingham covariates.

**Comparing statistical models with cross-validation and area under the curve (AUC) statistic**

In evaluating our statistical models, we used "area under the curve" (AUC) from our receiver operating characteristic (ROC) curves. Each statistical model was evaluated with 5-fold cross-validation. For those unfamiliar with cross-validation, five-fold cross-validation means that we partition the subjects into 5 non-overlapping "folds" of approximately equal size. We then fit 5 models, using 4 of the 5 folds as the "training" set for each fit. For each fit, we then test the model with fold that wasn't used in the corresponding training set. For example, for five-fold CV, we first omit fold #1 and fit the model using folds 2,3,4, and 5 together. We then test the model using fold #1. We then fit a model with folds 1,3,4, and 5 together and test it with fold #2. We continue this procedure until all five possible models are fitted. For each of the five models, we created a received operating characteristic (ROC) curve and evaluated the area under the curve using the pROC R package.[5]

A ROC curve is a plot of sensitivity against specificity. We created the ROC curve (~**??**)

add label to figure with ROC curve, after the caption for that figure.

by varying the threshold cut point for our classifier and seeing how distinct cut points impact our specificity and sensitivity as measured on the test set.

We used gradient boosting models, as implemented in the R package gbm, to identify the most important variables for each outcome. We noted that these covariates differed by outcome variable. We fit models that consisted of the top one, top two, top three, etc. covariates for each of our outcome variables. We performed five-fold cross-validation to get the AUC from the ROC curves. For example, in Figure 2, we have plotted the mean of the five AUCs for each model (with each of the six outcome variables). Consider

[5] Ibid.

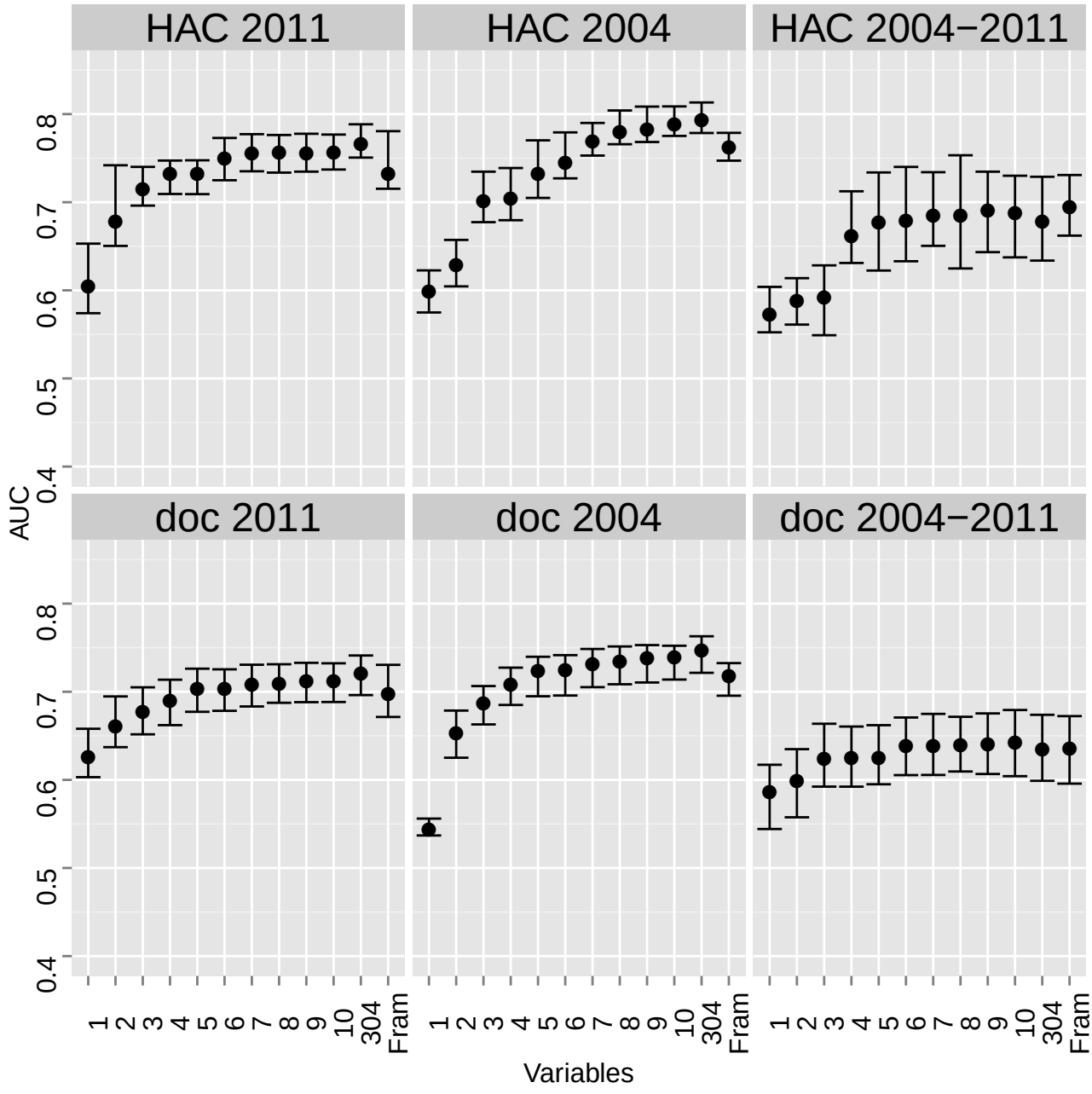


Figure 2: Comparing AUCs for GBM-based analyses.

the upper left plot, which displays the AUCs for the models with outcome variable HAC2011. On the vertical axis, we have AUC; number of variables in the model is plotted on the x-axis. The leftmost point represents the mean AUC when we use only the most important variable (for the HAC2011 outcome). The error bars extend to the maximum and minimum of the five AUCs for this model. The point with x=2 is the mean AUC for the model with the two most important variables for HAC2011 outcome. Note that the first eleven models in this figure are nested. The last model on the upper left plot (all the way to the righthand side of the plot) is that obtained by using the six Framingham covariates. We created the other five plots in 2 in an analogous fashion, by examining the most important covariates for each outcome variable.

We also investigated the role of tree pruning in single-tree models. An important advantage of the CART algorithm, as implemented in the R packages rpart, is that we have a more interpretable result, a single tree, rather than the ensemble product of the GBM algorithm.

## Discussion

## Future directions

We would like to investigate other tree-based methods, including those developed by Wei-Yin Loh's research team. The absence of R implementations of Loh's algorithms prevented us from including them in our current analysis. We have begun the process of writing code to implement Loh's algorithms in R, but they are not yet ready for use.

explain why we want Loh's algorithms. Might mention why they could be useful in this project.

A major reason for using Loh's algorithms, such as GUIDE, is their unbiasedness.[6] CART, the algorithm that's implemented in the rpart package, has an intrinsic selection bias that favors selection of categorical variables with more discrete values. GUIDE, on the other hand, avoids this selection bias. In the current scenario, because most of our variables are either continuous or binary, we're not selection bias is likely to be a major problem, but we'd still like to compare GUIDE results with those of CART.

Given the relatively low cost of acquiring genomics data, one possible future direction is to acquire genomics data for a subset of study subjects. For example, SNP genotype data from each subject may enable us to further discriminate those subjects that are at high risk for a CAD event. In some cases, we may be able to use cheek swabs as sources of DNA, which would enable data collection by mail. Such knowledge of genetic risk factors, when coupled with non-genetic risk factors, may be translated into the proposed intervention program, for example, by promoting healthy diet and physical activity among those at greatest risk.

[6] Loh, "Classification and Regression Trees."

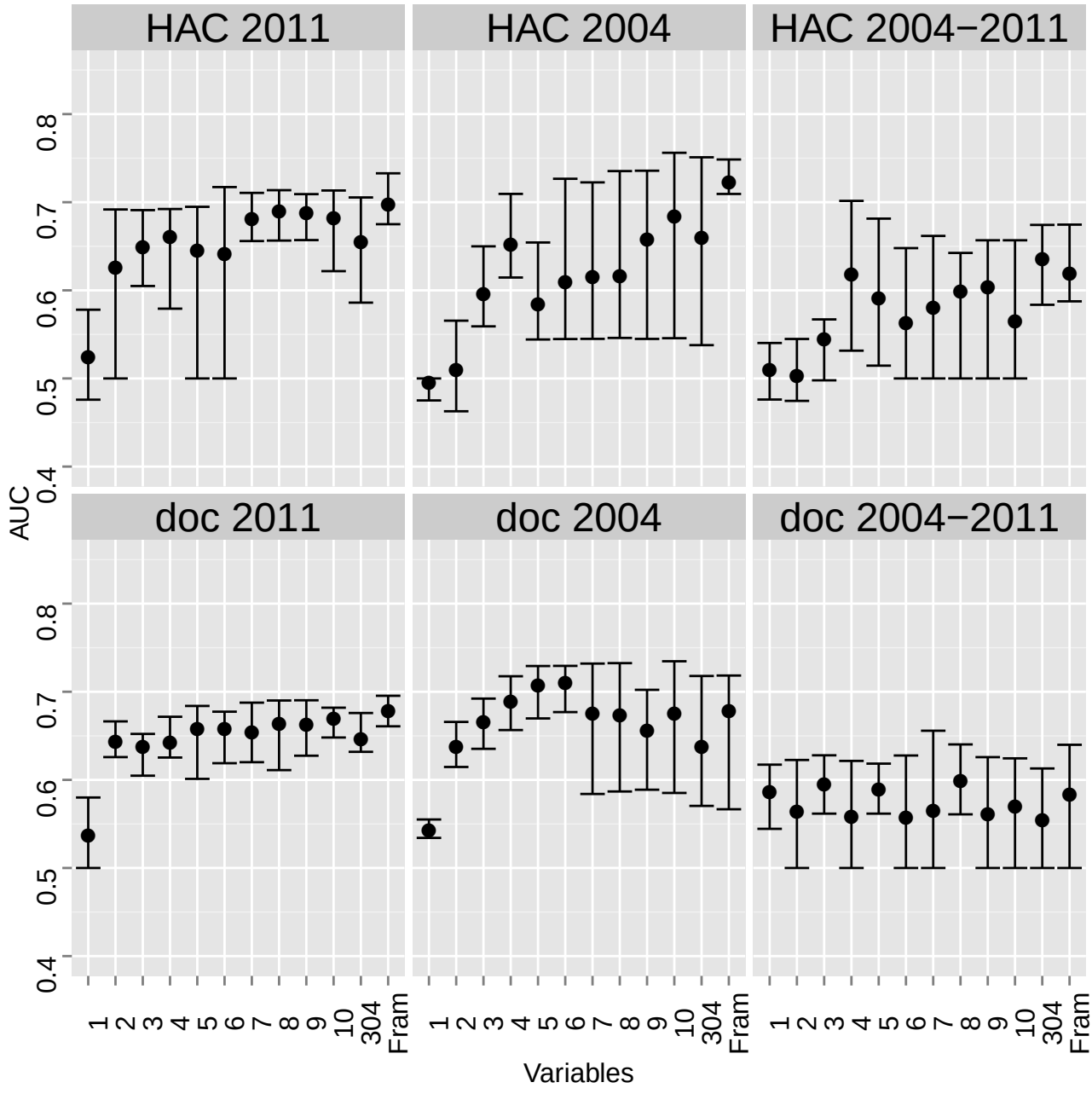


Figure 3: Comparing AUCs for CART-based analyses, based on CART complexity parameter.

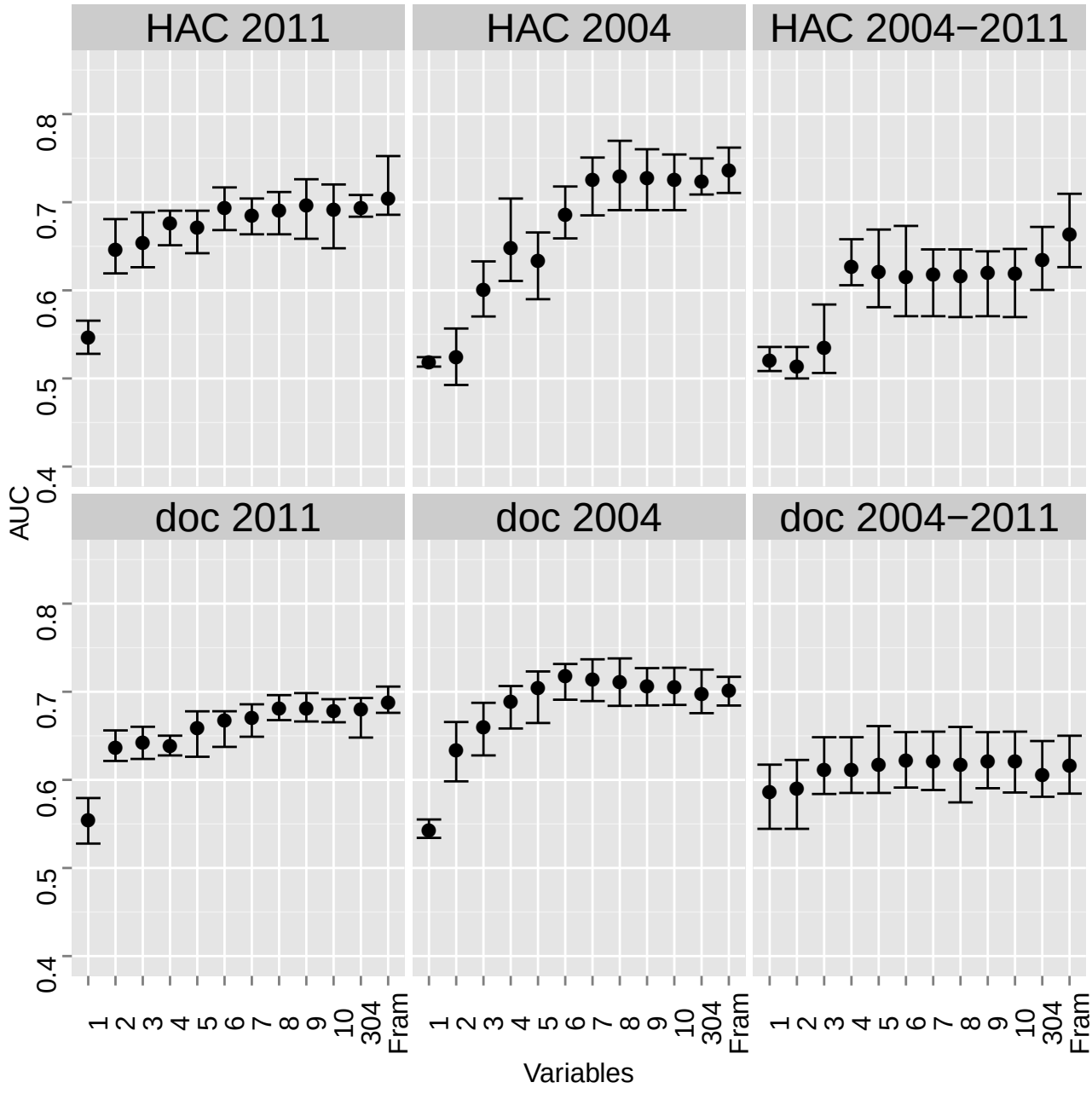


Figure 4: Comparing AUCs for CART-based analyses, based on deviance criterion.

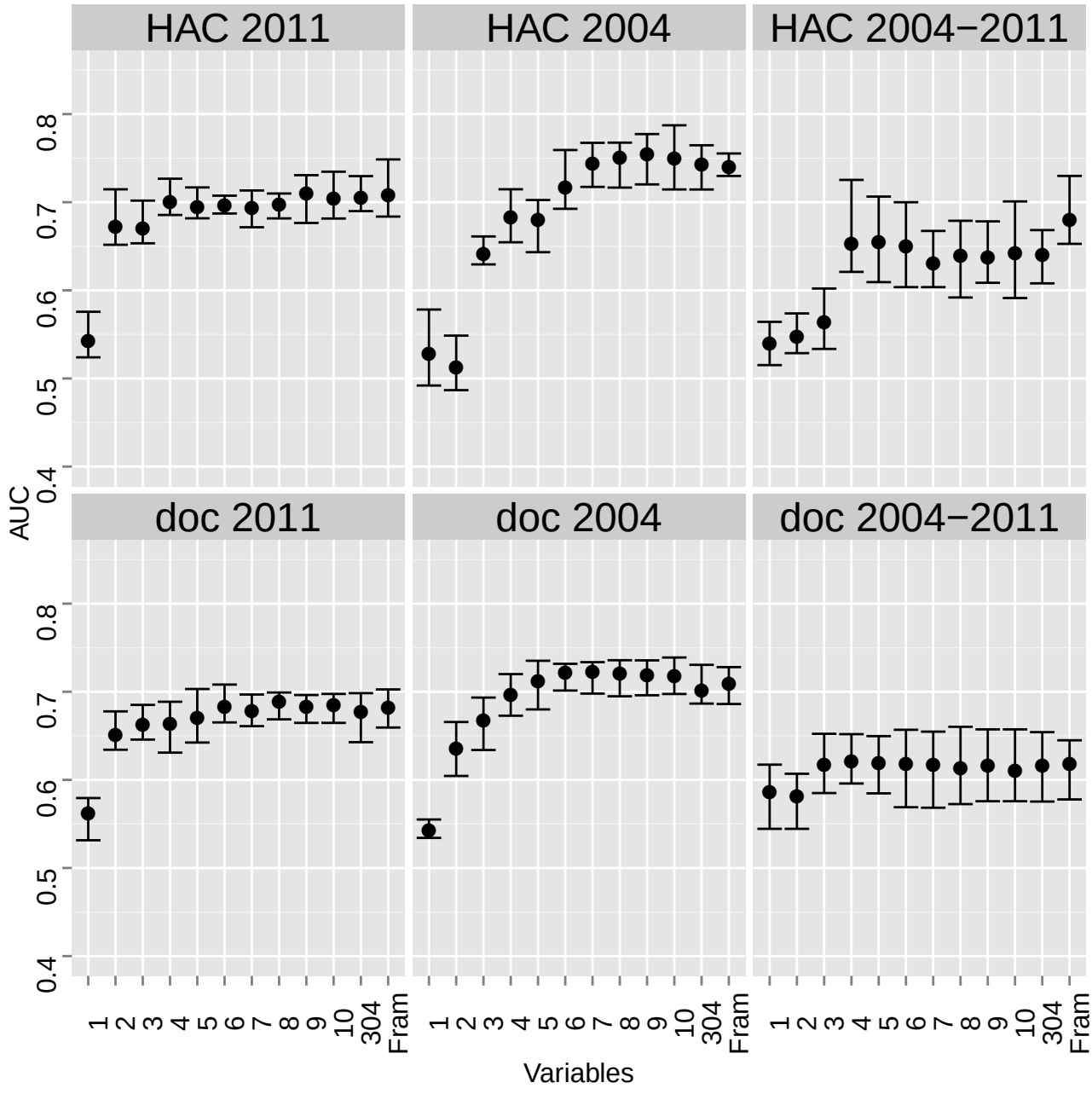


Figure 5: Comparing AUCs for CART-based analyses, based on min-split criterion.

## Appendix A: Supplementary materials

## Appendix B: Questions for client

1. What do you intend to do with results from this study?
2. How important is prediction of future MI to your scientific goals? There may be a tradeoff between prediction and the ability to quantify variable effects (for example does smoking increase or decrease risk? By how much? Does the effect vary across population subgroups?).
3. Why are there so many missing values (NA) for the HA2011 & HA2004 variables? Did these subjects not respond to that question? Did they respond to the survey at all?
4. Which variables do you think are the most meaningful?
5. Is incidence of MI between 2004 and 2011 a meaningful outcome? We could try to ascertain who had a MI during that interval (from among the people who had no MI history in 2004)
6. We're considering focusing on established risk factors such as those that the Framingham study identified. Is this reasonable? Or do you want to try to identify novel risk factors?
7. Are the "created" variables HAC2011 and HAC2004 better than the unadjusted versions? Is there any reason to do separate analyses for 2011 and 2004? Could we just code "yes" for a yes in any year, otherwise use 2011 value.
   Why do some responses switch from "yes" in 2004 to "no" in 2011—is this because the question specifically refers to heart attacks or diagnoses within the last 20 years?
8. Should any conditions be excluded as covariates because they can occur concurrently with heart disease (e.g. diabetes, stroke, high blood pressure, high cholesterol)? In other words, is there more interest in leading indicators?
9. Is there interest in separating the effects of covariates gathered in multiple years (e.g. highchol2004, highchol2011)? Should these be combined into a single measure? Should 2011 effect be excluded when modeling a 2004 response?
10. I noticed some ordinal variables (e.g. education level) are coded as integers. Is this consistent for all ordinal variables? Are there any non-ordinal categorical variables coded as integers?

## Appendix C: Computing code

```
## ----"setup", echo=FALSE----------------------------------------------------
rm(list=ls())

library(knitr)
opts_chunk$set(cache=FALSE, echo=FALSE, results= 'hide', message = FALSE, warning=FALSE, tidy=TRU


## ----"readdat"--------------------------------------------------------------
library(tidyr)
library(dplyr)
library(stringr)
library(lubridate)
library(ggplot2)
library(gbm)
wls<- tbl_df(read.csv("WLS2.csv"))


## ----HAC2011responders------------------------------------------------------
wls_ha2011responders <- wls[!is.na(wls$HAC2011),]



## ----logistic_framingham----------------------------------------------------

glm_wrap<- function(formula, data= wls, family="binomial"
                    )
    {
    mylogit <- glm(formula, data = data, family = family)
    # convert logits to probabilities
    pred_prob <- expit(predict(mylogit))
    df<- data.frame(mylogit$model[,1], pred_prob)
    names(df)<- c("outcome", "predicted_probability")
    return(df)
    }

     expit <- function(l)
  {
  exp(l)/(1+exp(l))
  }
pp_df<- glm_wrap(HAC2011 ~ sex + age + highchol2011 + smokever2011 + diabetes2011 + highbp2011)


## ----pROC-------------------------------------------------------------------
library(pROC)

plot.roc_wrap <- function(out_df # a dataframe outputted from glm_wrap()
                          )
```

```
    {
    plot.roc(out_df$outcome, out_df$predicted_probability, print.auc=TRUE, ci=FALSE)
    }



## ----plot-hac2011, fig.cap= "Receiver operating characteristic curve for the logistic regressio
plot.roc_wrap(glm_wrap(HAC2011 ~ sex + age + highchol2011 + smokever2011 + diabetes2011 + highbp2


## ----plot-doc2011, fig.show='hide'-------------------------------------
plot.roc_wrap(glm_wrap(doc2011 ~ sex + age + highchol2011 + smokever2011 + diabetes2011 + highbp2


## ----plot-hac2004, fig.show='hide'-------------------------------------
plot.roc_wrap(glm_wrap(HAC2004 ~ sex + age + highchol2011 + smokever2011 + diabetes2011 + highbp2


## ----plot-doc2004, fig.show='hide'-------------------------------------
plot.roc_wrap(glm_wrap(doc2004 ~ sex + age + highchol2011 + smokever2011 + diabetes2011 + highbp2



## ----subdivide_sex-----------------------------------------------------
wls_fem<- wls[wls$sex == "Female",]
wls_mal<- wls[wls$sex == "Male",]


## ----plot-hac2011-males, fig.show='hide'-------------------------------
plot.roc_wrap(glm_wrap(HAC2011 ~ age + highchol2011 + smokever2011 + diabetes2011 + highbp2011, d
plot.roc_wrap(glm_wrap(doc2011 ~ age + highchol2011 + smokever2011 + diabetes2011 + highbp2011, d
plot.roc_wrap(glm_wrap(HAC2004 ~ age + highchol2011 + smokever2011 + diabetes2011 + highbp2011, d
plot.roc_wrap(glm_wrap(doc2004 ~ age + highchol2011 + smokever2011 + diabetes2011 + highbp2011, d



## ----plot-hac2011-females, fig.show='hide'-----------------------------
plot.roc_wrap(glm_wrap(HAC2011 ~ age + highchol2011 + smokever2011 + diabetes2011 + highbp2011, d
plot.roc_wrap(glm_wrap(doc2011 ~ age + highchol2011 + smokever2011 + diabetes2011 + highbp2011, d
plot.roc_wrap(glm_wrap(HAC2004 ~ age + highchol2011 + smokever2011 + diabetes2011 + highbp2011, d
plot.roc_wrap(glm_wrap(doc2004 ~ age + highchol2011 + smokever2011 + diabetes2011 + highbp2011, d



## ----subdivide-diabetes2004, fig.show='hide'---------------------------
wls_dm04<- wls[wls$diabetes2004 == "Yes",]
wls_nodm04<- wls[wls$diabetes2004 == "No",]
plot.roc_wrap(glm_wrap(HAC2011 ~ sex + age + highchol2011 + smokever2011 + highbp2011, data=wls_n
```

```
## ------------------------------------------------------------------------
# write a function that 1. partitions wls by a binary variable and outputs two data subsets as a
subdivide <- function(data = wls, indicator = sex)
    {
    out <- list(wls[indicator,], wls[!indicator,])
    return(out)
    }



## ----fivefoldcv----------------------------------------------------------
df_in <- with(wls, data.frame(sex, age, highchol2011, smokever2011, diabetes2011, highbp2011))
source("glmcv.R")
glmcv(x=df_in, y=wls$HAC2011)
```

```
require(caret)
require(car)
require(ggplot2)
require(pROC)
require(gbm)
require(rpart)
require(rpart.plot)

## Framingham variables
framvar11 <- c("sex", "age", "highchol2011", "smokever2011", "diabetes2011",
    "highbp2011")
framvar04 <- c("sex", "age", "highchol2004", "smokever2004", "diabetes2004",
    "highbp2004")

## append missing value indicators to covariates 'x'
dat <- read.csv("/mnt/DATA/Main/Spring 2015/998/Proj2/docs/WLS2.csv")
x1 <- dat[, !colnames(dat) %in% c("HA2011", "HA2004", "HAC2011", "HAC2004",
    "doc2011", "doc2004")]
miss_pct <- sapply(colnames(x1), function(nam) mean(is.na(dat[, nam])))
x2 <- data.frame((is.na(x1[, miss_pct > 0.1]) * 1))
names(x2) <- paste0(names(x1[miss_pct > 0.1]), "_missid")
x <- data.frame(x1, x2)  # appended with a missing value indicator for each variable
id_11 <- grepl("2011", names(x))
x04 <- x[, !id_11]  # excluding 2011 covariates
id1_11 <- grepl("2011", names(x1))
x104 <- x1[, !id1_11]  # no missing value indicators, excluding 2011 covariates

## create cross validation folds for response variables Note gbm requires
## integer response in {0,1}
h11 <- Recode(dat$HAC2011, "'Yes'=1; 'No'=0", as.factor = F)
h04 <- Recode(dat$HAC2004, "'Yes'=1; 'No'=0", as.factor = F)
d11 <- Recode(dat$doc2011, "'Yes'=1; 'No'=0", as.factor = F)
d04 <- Recode(dat$doc2004, "'Yes'=1; 'No'=0", as.factor = F)
h0411 <- rep(0, length(h11))
```

```
h0411[is.na(h04 + h11) | h04 == 1] <- NA
h0411[h04 == 0 & h11 == 1] <- 1
d0411 <- rep(0, length(d11))
d0411[is.na(d04 + d11) | d04 == 1] <- NA
d0411[d04 == 0 & d11 == 1] <- 1

set.seed(9881829)
folds_h11_temp <- createFolds(which(!is.na(h11)), k = 5, list = T, returnTrain = F)
folds_h11 <- lapply(folds_h11_temp, function(x) which(!is.na(h11))[x])
folds_h04_temp <- createFolds(which(!is.na(h04)), k = 5, list = T, returnTrain = F)
folds_h04 <- lapply(folds_h04_temp, function(x) which(!is.na(h04))[x])
folds_h0411_temp <- createFolds(which(!is.na(h0411)), k = 5, list = T, returnTrain = F)
folds_h0411 <- lapply(folds_h0411_temp, function(x) which(!is.na(h0411))[x])
folds_d11_temp <- createFolds(which(!is.na(d11)), k = 5, list = T, returnTrain = F)
folds_d11 <- lapply(folds_d11_temp, function(x) which(!is.na(d11))[x])
folds_d04_temp <- createFolds(which(!is.na(d04)), k = 5, list = T, returnTrain = F)
folds_d04 <- lapply(folds_d04_temp, function(x) which(!is.na(d04))[x])
folds_d0411_temp <- createFolds(which(!is.na(d0411)), k = 5, list = T, returnTrain = F)
folds_d0411 <- lapply(folds_d0411_temp, function(x) which(!is.na(d0411))[x])
rm(folds_h11_temp, folds_h04_temp, folds_h0411_temp, folds_d11_temp, folds_d04_temp,
    folds_d0411_temp)

#### Test for optimal GBM tuning parameters

## function to get cross validation deviance, modified deviance, mse,
## misclassification rate, and auc
gbmCV <- function(x, y, folds, tree_inc, dist = "adaboost", id = 1, bf = 0.5,
    sh = 0.001, singlefold = F) {

    getDev <- function(p, y) -mean(ifelse(y == 1, log(p), log(1 - p)))
    getDevmod <- function(p, y) getDev(pmin(pmax(p, 0.001), 0.999), y)
    getMse <- function(p, y) mean((y - p)^2)
    getMisclass <- function(p, y) mean(abs((p > 0.5) - y))
    getAUC <- function(p, y) auc(y, p)

    gbmInit2 <- function(y, x, tree_inc) {
        tree_ct <- tree_inc
        gbm_fit <- gbm(y ~ ., data = x, distribution = dist, n.tree = tree_inc,
            shrinkage = sh, interaction.depth = id, bag.fraction = bf)
        best_tree <- gbm.perf(gbm_fit, method = "OOB")
        while (best_tree/tree_ct > 0.99) {
            tree_ct <- tree_ct + tree_inc
            gbm_fit <- gbm.more(gbm_fit, n.new.trees = tree_inc)
            best_tree <- gbm.perf(gbm_fit, method = "OOB")
        }
        list(gbm_fit = gbm_fit, best_tree = best_tree)
    }

    best_tree <- vector()
```

```
    dev <- vector()
    devmod <- vector()
    mse <- vector()
    misclass <- vector()
    auc1 <- vector()
    cv_num <- if (singlefold)
        1 else length(folds)
    for (i in 1:cv_num) {
        train_id <- Reduce(union, folds[-i])
        test_id <- folds[[i]]
        xtrain <- x[train_id, , drop = F]
        ytrain <- y[train_id]
        xtest <- x[test_id, , drop = F]
        ytest <- y[test_id]
        init <- gbmInit2(ytrain, xtrain, tree_inc)
        gbm_fit <- init$gbm_fit
        best_tree[i] <- init$best_tree
        ## predicted probability of class 1
        pred <- predict(gbm_fit, newdata = xtest, n.trees = best_tree[i], type = "response")
        dev[i] <- getDev(pred, ytest)
        devmod[i] <- getDevmod(pred, ytest)
        mse[i] <- getMse(pred, ytest)
        misclass[i] <- getMisclass(pred, ytest)
        auc1[i] <- getAUC(pred, ytest)
    }

    cbind(dev, devmod, mse, misclass, auc = auc1, best_tree)
}  # end gbmCV

## 1-fold gbm tuning parameter tests (excluding missing value indicators to
## save time)
par <- rbind(data.frame(dist = "adaboost", id = 1, bf = 0.5, sh = 0.01), data.frame(dist = "berno
    id = 1, bf = 0.5, sh = 0.01), data.frame(dist = "adaboost", id = 3, bf = 0.5,
    sh = 0.01), data.frame(dist = "bernoulli", id = 3, bf = 0.5, sh = 0.01),
    data.frame(dist = "adaboost", id = 5, bf = 0.5, sh = 0.01), data.frame(dist = "bernoulli",
        id = 5, bf = 0.5, sh = 0.01), data.frame(dist = "adaboost", id = 7,
        bf = 0.5, sh = 0.01), data.frame(dist = "bernoulli", id = 7, bf = 0.5,
        sh = 0.01), data.frame(dist = "adaboost", id = 1, bf = 0.3, sh = 0.01),
    data.frame(dist = "bernoulli", id = 1, bf = 0.3, sh = 0.01), data.frame(dist = "adaboost",
        id = 1, bf = 0.7, sh = 0.01), data.frame(dist = "bernoulli", id = 1,
        bf = 0.7, sh = 0.01), data.frame(dist = "adaboost", id = 1, bf = 0.5,
        sh = 0.001), data.frame(dist = "bernoulli", id = 1, bf = 0.5, sh = 0.001))
# set.seed(95709963) gbmcvtest_h11 <- t(mapply(dist=as.character(par$dist),
# id=par$id, bf=par$bf, sh=par$sh, function(dist, id, bf, sh) gbmCV(x1, h11,
# folds_h11, tree_inc=1000, dist=dist, id=id, bf=bf, sh=sh, singlefold=T)))
# colnames(gbmcvtest_h11) <-
# c('dev','devmod','mse','misclass','auc','best_tree') gbmcvtest_d11 <-
# t(mapply(dist=as.character(par$dist), id=par$id, bf=par$bf, sh=par$sh,
# function(dist, id, bf, sh) gbmCV(x1, d11, folds_d11, tree_inc=1000,
```

```
# dist=dist, id=id, bf=bf, sh=sh, singlefold=T))) colnames(gbmcvtest_d11) <-
# c('dev','devmod','mse','misclass','auc','best_tree') save(gbmcvtest_h11,
# gbmcvtest_d11, file='/mnt/DATA/Main/Spring
# 2015/998/Proj2/models/gbmcvtest.rda')
load("/mnt/DATA/Main/Spring 2015/998/Proj2/models/gbmcvtest.rda")
cbind(par, gbmcvtest_h11)
cbind(par, gbmcvtest_d11)


#### Fit GBM models using optimal tuning parameters to get Relative Importance
#### scores

## function to grow incremental trees until the out of bag optimum number is
## reached
gbmInit <- function(y, x, folds, tree_inc = 1000, sh = 0.001) {
    id <- Reduce(union, folds)
    xfit <- x[id, ]
    yfit <- y[id]
    ntree <- tree_inc
    gbm_fit <- gbm.fit(xfit, yfit, distribution = "adaboost", n.tree = tree_inc,
        shrinkage = sh)
    while (gbm.perf(gbm_fit, method = "OOB") == ntree) {
        ntree <- ntree + tree_inc
        gbm_fit <- gbm.more(gbm_fit, n.new.trees = tree_inc)
    }
    gbm_fit
}

# set.seed(96633) gbm_h11b <- gbmInit(h11, x, folds_h11, 1000, .01) gbm_h04b
# <- gbmInit(h04, x04, folds_h04, 1000, .01) gbm_h0411b <- gbmInit(h0411,
# x04, folds_h0411, 1000, .01) gbm_d11b <- gbmInit(d11, x, folds_d11, 1000,
# .01) gbm_d04b <- gbmInit(d04, x04, folds_d04, 1000, .01) gbm_d0411b <-
# gbmInit(d0411, x04, folds_d0411, 1000, .01) save(gbm_h11b, gbm_h04b,
# gbm_h0411b, gbm_d11b, gbm_d04b, gbm_d0411b, file='/mnt/DATA/Main/Spring
# 2015/998/Proj2/models/gbm-initialb.rda')
load("/mnt/DATA/Main/Spring 2015/998/Proj2/models/gbm-initialb.rda")

best_h11b <- gbm.perf(gbm_h11b, method = "OOB")
best_h04b <- gbm.perf(gbm_h04b, method = "OOB")
best_h0411b <- gbm.perf(gbm_h0411b, method = "OOB")
best_d11b <- gbm.perf(gbm_d11b, method = "OOB")
best_d04b <- gbm.perf(gbm_d04b, method = "OOB")
best_d0411b <- gbm.perf(gbm_d0411b, method = "OOB")

summary(gbm_h11b, n.trees = best_h11b)[summary(gbm_h11b, n.trees = best_h11b)$rel.inf >
    0, ]
summary(gbm_h04b, n.trees = best_h04b)[summary(gbm_h04b, n.trees = best_h04b)$rel.inf >
    0, ]
summary(gbm_h0411b, n.trees = best_h0411b)[summary(gbm_h0411b, n.trees = best_h0411b)$rel.inf >
```

```
    0, ]
summary(gbm_d11b, n.trees = best_d11b)[summary(gbm_d11b, n.trees = best_d11b)$rel.inf >
    0, ]
summary(gbm_d04b, n.trees = best_d04b)[summary(gbm_d04b, n.trees = best_d04b)$rel.inf >
    0, ]
summary(gbm_d0411b, n.trees = best_d0411b)[summary(gbm_d0411b, n.trees = best_d0411b)$rel.inf >
    0, ]

#### GBM 5-fold cross validation

## GBM 5-fold cross validation--all variables gbmcv_h11 <- gbmCV(x1, h11,
## folds_h11, tree_inc=1000, sh=.01) gbmcv_h04 <- gbmCV(x104, h04, folds_h04,
## tree_inc=1000, sh=.01) gbmcv_h0411 <- gbmCV(x104, h0411, folds_h0411,
## tree_inc=1000, sh=.01) gbmcv_d11 <- gbmCV(x1, d11, folds_d11,
## tree_inc=1000, sh=.01) gbmcv_d04 <- gbmCV(x104, d04, folds_d04,
## tree_inc=1000, sh=.01) gbmcv_d0411 <- gbmCV(x104, d0411, folds_d0411,
## tree_inc=1000, sh=.01) save(gbmcv_h11, gbmcv_h04, gbmcv_h0411, gbmcv_d11,
## gbmcv_d04, gbmcv_d0411, file='/mnt/DATA/Main/Spring
## 2015/998/Proj2/models/gbmcv.rda')
load("/mnt/DATA/Main/Spring 2015/998/Proj2/models/gbmcv.rda")
gbmcv_h11
gbmcv_h04
gbmcv_h0411
gbmcv_d11
gbmcv_d04
gbmcv_d0411

## GBM 5-fold gbm tests--reduced variable list set.seed(3766716) gbmcv_h11r
## <- lapply(1:10, function(num_var) { var <-
## as.character(summary(gbm_h11b)[1:num_var,1]) gbmCV(x[,var,drop=F], h11,
## folds_h11, tree_inc=1000, sh=.01) }) gbmcv_h04r <- lapply(1:10,
## function(num_var) { var <- as.character(summary(gbm_h04b)[1:num_var,1])
## gbmCV(x[,var,drop=F], h04, folds_h04, tree_inc=1000, sh=.01) })
## gbmcv_h0411r <- lapply(1:10, function(num_var) { var <-
## as.character(summary(gbm_h0411b)[1:num_var,1]) gbmCV(x[,var,drop=F],
## h0411, folds_h0411, tree_inc=1000, sh=.01) }) gbmcv_d11r <- lapply(1:10,
## function(num_var) { var <- as.character(summary(gbm_d11b)[1:num_var,1])
## gbmCV(x[,var,drop=F], d11, folds_d11, tree_inc=1000, sh=.01) }) gbmcv_d04r
## <- lapply(1:10, function(num_var) { var <-
## as.character(summary(gbm_d04b)[1:num_var,1]) gbmCV(x[,var,drop=F], d04,
## folds_d04, tree_inc=1000, sh=.01) }) gbmcv_d0411r <- lapply(1:10,
## function(num_var) { var <- as.character(summary(gbm_d0411b)[1:num_var,1])
## gbmCV(x[,var,drop=F], d0411, folds_d0411, tree_inc=1000, sh=.01) })
## save(gbmcv_h11r, gbmcv_h04r, gbmcv_h0411r, gbmcv_d11r, gbmcv_d04r,
## gbmcv_d0411r, file='/mnt/DATA/Main/Spring
## 2015/998/Proj2/models/gbmcvr.rda')
load("/mnt/DATA/Main/Spring 2015/998/Proj2/models/gbmcvr.rda")
gbmcv_h11r
gbmcv_h04r
```

```
gbmcv_h0411r
gbmcv_d11r
gbmcv_d04r
gbmcv_d0411r

## GBM 5-fold cross validation--Framingham variables gbmcv_h11fram <-
## gbmCV(x[,framvar11], h11, folds_h11, tree_inc=1000, sh=.01) gbmcv_h04fram
## <- gbmCV(x[,framvar04], h04, folds_h04, tree_inc=1000, sh=.01)
## gbmcv_h0411fram <- gbmCV(x[,framvar04], h0411, folds_h0411, tree_inc=1000,
## sh=.01) gbmcv_d11fram <- gbmCV(x[,framvar11], d11, folds_d11,
## tree_inc=1000, sh=.01) gbmcv_d04fram <- gbmCV(x[,framvar04], d04,
## folds_d04, tree_inc=1000, sh=.01) gbmcv_d0411fram <- gbmCV(x[,framvar04],
## d0411, folds_d0411, tree_inc=1000, sh=.01) save(gbmcv_h11fram,
## gbmcv_h04fram, gbmcv_h0411fram, gbmcv_d11fram, gbmcv_d04fram,
## gbmcv_d0411fram, file='/mnt/DATA/Main/Spring
## 2015/998/Proj2/models/gbmcvfram.rda')
load("/mnt/DATA/Main/Spring 2015/998/Proj2/models/gbmcvfram.rda")
gbmcv_h11fram
gbmcv_h04fram
gbmcv_h0411fram
gbmcv_d11fram
gbmcv_d04fram
gbmcv_d0411fram

## GBM plot cross validation results
getMean <- function(x) mean(x[, "auc"])
getMin <- function(x) min(x[, "auc"])
getMax <- function(x) max(x[, "auc"])
gbm_plotdata <- data.frame(resp = factor(rep(c("HAC 2011", "HAC 2004", "HAC 2004-2011",
    "doc 2011", "doc 2004", "doc 2004-2011"), each = 12), levels = c("HAC 2011",
    "HAC 2004", "HAC 2004-2011", "doc 2011", "doc 2004", "doc 2004-2011")),
    Variables = factor(rep(c(1:10, 304, "Fram"), 6), levels = c(1:10, 304, "Fram")),
    AUC = c(sapply(gbmcv_h11r, getMean), getMean(gbmcv_h11), getMean(gbmcv_h11fram),
        sapply(gbmcv_h04r, getMean), getMean(gbmcv_h04), getMean(gbmcv_h04fram),
        sapply(gbmcv_h0411r, getMean), getMean(gbmcv_h0411), getMean(gbmcv_h0411fram),
        sapply(gbmcv_d11r, getMean), getMean(gbmcv_d11), getMean(gbmcv_d11fram),
        sapply(gbmcv_d04r, getMean), getMean(gbmcv_d04), getMean(gbmcv_d04fram),
        sapply(gbmcv_d0411r, getMean), getMean(gbmcv_d0411), getMean(gbmcv_d0411fram)),
    min = c(sapply(gbmcv_h11r, getMin), getMin(gbmcv_h11), getMin(gbmcv_h11fram),
        sapply(gbmcv_h04r, getMin), getMin(gbmcv_h04), getMin(gbmcv_h04fram),
        sapply(gbmcv_h0411r, getMin), getMin(gbmcv_h0411), getMin(gbmcv_h0411fram),
        sapply(gbmcv_d11r, getMin), getMin(gbmcv_d11), getMin(gbmcv_d11fram),
        sapply(gbmcv_d04r, getMin), getMin(gbmcv_d04), getMin(gbmcv_d04fram),
        sapply(gbmcv_d0411r, getMin), getMin(gbmcv_d0411), getMin(gbmcv_d0411fram)),
    max = c(sapply(gbmcv_h11r, getMax), getMax(gbmcv_h11), getMax(gbmcv_h11fram),
        sapply(gbmcv_h04r, getMax), getMax(gbmcv_h04), getMax(gbmcv_h04fram),
        sapply(gbmcv_h0411r, getMax), getMax(gbmcv_h0411), getMax(gbmcv_h0411fram),
        sapply(gbmcv_d11r, getMax), getMax(gbmcv_d11), getMax(gbmcv_d11fram),
        sapply(gbmcv_d04r, getMax), getMax(gbmcv_d04), getMax(gbmcv_d04fram),
```

```
        sapply(gbmcv_d0411r, getMax), getMax(gbmcv_d0411), getMax(gbmcv_d0411fram)))

# pdf('/mnt/DATA/Main/Spring 2015/998/Proj2/plots/gbm all cv.pdf')
# ggplot(gbm_plotdata, aes(y=AUC, x=Variables)) + geom_point(size=3) +
# geom_errorbar(aes(y=AUC, ymin=min, ymax=max)) + facet_wrap(~resp) +
# theme(strip.text=element_text(size=18), axis.text=element_text(angle=90,
# color='black', size=12), axis.title=element_text(size=12)) + ylim(.4,.85)
# dev.off()


#### RPART Classification Trees

## modified rpart.prune function to allow deviance and minsplit based pruning
## method can be 'cp', 'dev_quantile', 'minsplit' prunes values strictly <
## pruneval (not <= as in prune.rpart)
myprune.rpart <- function(tree, method, prune_val, ...) {
    ff <- tree$frame
    id <- as.integer(row.names(ff))
    ## my lines
    if (method == "cp") {
        toss <- id[ff$complexity < prune_val & ff$var != "<leaf>"]  #not a leaf
    } else if (method == "dev") {
        ss <- tree$splits  # according to rpart.object doc, 'improve' refers to deviance
        ss_id <- cumsum(ff$var != "<leaf>") + c(0, head(cumsum(ff$ncompete +
            ff$nsurrogate), -1))
        imp <- ss[ss_id, "improve"]
        imp[ff$var == "<leaf>"] <- NA  # NA for terminal nodes
        # dev_cutoff <- quantile(imp, prune_val, na.rm=T) toss <- id[imp <
        # dev_cutoff & ff$var != '<leaf>'] #not a leaf
        toss <- id[imp < prune_val & ff$var != "<leaf>"]  #not a leaf
    } else if (method == "minsplit") {
        toss <- id[ff$n < prune_val & ff$var != "<leaf>"]
    }
    if (length(toss) == 0L)
        return(tree)  # all the tree is retained
    newx <- snip.rpart(tree, toss)
    ## Now cut down the CP table temp <- pmax(tree$cptable[, 1L], cp) keep <-
    ## match(unique(temp), temp) newx$cptable <- tree$cptable[keep, , drop =
    ## FALSE] newx$cptable[max(keep), 1L] <- cp # Reset the variable importance
    ## newx$variable.importance <- importance(newx)
    newx$cptable <- NULL
    newx$variable.importance <- NULL
    newx
}

## function to determine optimal pruning parameter from cross validation
## Returns 3 lists (cp, dev, minsplit). Each list has length = # cross
## validation folds.  Each list element is a table with dev, devmod, mse,
## misclass, auc for each pruning value no cp_vals needed: uses the values
```

```
## from $cptable
rpartPruneCV <- function(x, y, folds, split = "gini", cp = -1, cp_quantiles = seq(0,
    1, 0.01), dev_quantiles = seq(0, 1, 0.01), minsplit_quantiles = seq(0, 1,
    0.01)) {

    dev <- function(p, y) -mean(ifelse(y == 1, log(p), log(1 - p)))
    devmod <- function(p, y) dev(pmin(pmax(p, 0.001), 0.999), y)
    mse <- function(p, y) mean((y - p)^2)
    misclass <- function(p, y) mean(abs((p > 0.5) - y))
    getAUC <- function(p, y) auc(y, p)

    cv_fun <- function(rpart_fit, method, prune_val) {
        rpart_prune <- myprune.rpart(rpart_fit, method = method, prune_val = prune_val)
        pred <- predict(rpart_prune, newdata = xtest)[, 2]
        dev <- dev(pred, ytest)
        devmod <- devmod(pred, ytest)
        mse <- mse(pred, ytest)
        misclass <- misclass(pred, ytest)
        auc <- getAUC(pred, ytest)
        c(dev = dev, devmod = devmod, mse = mse, misclass = misclass, auc = auc)
    }

    rpart_trainfit <- list()
    for (i in 1:length(folds)) {
        train_id <- Reduce(union, folds[-i])
        xtrain <- x[train_id, , drop = F]
        ytrain <- y[train_id]
        rpart_trainfit[[i]] <- rpart(ytrain ~ ., data = xtrain, method = "class",
            parms = list(split = split), control = rpart.control(maxsurrogate = 100,
                cp = cp))
    }
    allcp <- unlist(lapply(rpart_trainfit, function(x) x$frame$complexity[x$frame$var !=
        "<leaf>"]))
    alldev <- unlist(lapply(rpart_trainfit, function(x) {
        ff <- x$frame
        ss <- x$splits  # according to rpart.object doc, 'improve' refers to deviance
        ss_id <- cumsum(ff$var != "<leaf>") + c(0, head(cumsum(ff$ncompete +
            ff$nsurrogate), -1))
        imp <- ss[ss_id, "improve"]
        imp[ff$var != "<leaf>"]
    }))
    allleafsize <- unlist(lapply(rpart_trainfit, function(x) x$frame$n[x$frame$var !=
        "<leaf>"]))
    cp_vals <- quantile(allcp, cp_quantiles)
    dev_vals <- quantile(alldev, dev_quantiles)
    minsplit_vals <- quantile(allleafsize, minsplit_quantiles)

    cv <- list()
    cv$cp <- list()
```

```
    cv$dev <- list()
    cv$minsplit <- list()
    for (i in 1:length(folds)) {
        test_id <- folds[[i]]
        xtest <- x[test_id, , drop = F]
        ytest <- y[test_id]
        test_loss <- t(sapply(cp_vals, function(v) cv_fun(rpart_trainfit[[i]],
            "cp", v)))
        cv$cp[[i]] <- cbind(cp_vals, test_loss)
        test_loss <- t(sapply(dev_vals, function(v) cv_fun(rpart_trainfit[[i]],
            "dev", v)))
        cv$dev[[i]] <- cbind(dev_vals, test_loss)
        test_loss <- t(sapply(minsplit_vals, function(v) cv_fun(rpart_trainfit[[i]],
            "minsplit", v)))
        cv$minsplit[[i]] <- cbind(minsplit_vals, test_loss)
    }
    cv
}

## function to get optimal pruning value from rpartcv table
getPruneValBest <- function(rpartcv, cv_method, prune_method, se_buffer = 1) {
    prune_vals <- sapply(rpartcv[[prune_method]], function(x) x[, 1])
    cv_loss <- sapply(rpartcv[[prune_method]], function(x) x[, cv_method])
    cv_loss_mean <- rowMeans(cv_loss) * -(cv_method == "auc")  # negative auc is a loss function
    cv_loss_se <- apply(cv_loss, 1, sd)/sqrt(ncol(cv_loss))
    cv_min <- min(cv_loss_mean)
    prune_vals[max(which(cv_loss_mean - se_buffer * cv_loss_se <= cv_min))]
}

## function to get cross validation deviance, modified deviance, mse,
## misclassification rate, auc uses rpartPruneCV to select pruning parameter
## by within-fold cross validation
## cv_method=c('none','dev','devmod','mse','misclass','auc')
rpartCV <- function(x, y, folds, cv_method = "auc", split = "gini", cp = -1,
    cp_quantiles = seq(0, 1, 0.01), dev_quantiles = seq(0, 1, 0.01), minsplit_quantiles = seq(0,
        1, 0.01)) {

    getDev <- function(p, y) -mean(ifelse(y == 1, log(p), log(1 - p)))
    getDevmod <- function(p, y) getDev(pmin(pmax(p, 0.001), 0.999), y)
    getMse <- function(p, y) mean((y - p)^2)
    getMisclass <- function(p, y) mean(abs((p > 0.5) - y))
    getAUC <- function(p, y) auc(y, p)

    cv <- list()
    cv$cp <- matrix(nrow = length(folds), ncol = 6)
    cv$dev <- matrix(nrow = length(folds), ncol = 6)
    cv$minsplit <- matrix(nrow = length(folds), ncol = 6)
    for (i in 1:length(folds)) {
        train_id <- Reduce(union, folds[-i])
```

```
    test_id <- folds[[i]]
    xtrain <- x[train_id, , drop = F]
    ytrain <- y[train_id]
    xtest <- x[test_id, , drop = F]
    ytest <- y[test_id]
    if (cv_method == "none") {
        prune_val_best <- cp
    } else {
        folds_tune <- createFolds(1:length(ytrain), k = 5, list = T, returnTrain = F)
        rpart_tune <- rpartPruneCV(xtrain, ytrain, folds_tune, split, cp,
            cp_quantiles, dev_quantiles, minsplit_quantiles)
        cp_prune_val_best <- getPruneValBest(rpart_tune, cv_method, "cp")
        dev_prune_val_best <- getPruneValBest(rpart_tune, cv_method, "dev")
        minsplit_prune_val_best <- getPruneValBest(rpart_tune, cv_method,
            "minsplit")
    }
    rpart_train <- rpart(ytrain ~ ., data = xtrain, method = "class", parms = list(split = sp
        control = rpart.control(maxsurrogate = 100, cp = cp))
    ## once for cp
    rpart_prune <- myprune.rpart(rpart_train, method = "cp", prune_val = cp_prune_val_best)
    pred <- predict(rpart_prune, newdata = xtest)[, 2]
    dev <- getDev(pred, ytest)
    devmod <- getDevmod(pred, ytest)
    mse <- getMse(pred, ytest)
    misclass <- getMisclass(pred, ytest)
    auc1 <- getAUC(pred, ytest)
    cv$cp[i, ] <- c(dev, devmod, mse, misclass, auc1, cp_prune_val_best)
    ## once for dev
    rpart_prune <- myprune.rpart(rpart_train, method = "dev", prune_val = dev_prune_val_best)
    pred <- predict(rpart_prune, newdata = xtest)[, 2]
    dev <- getDev(pred, ytest)
    devmod <- getDevmod(pred, ytest)
    mse <- getMse(pred, ytest)
    misclass <- getMisclass(pred, ytest)
    auc1 <- getAUC(pred, ytest)
    cv$dev[i, ] <- c(dev, devmod, mse, misclass, auc1, dev_prune_val_best)
    ## once for minsplit
    rpart_prune <- myprune.rpart(rpart_train, method = "minsplit", prune_val = minsplit_prune
    pred <- predict(rpart_prune, newdata = xtest)[, 2]
    dev <- getDev(pred, ytest)
    devmod <- getDevmod(pred, ytest)
    mse <- getMse(pred, ytest)
    misclass <- getMisclass(pred, ytest)
    auc1 <- getAUC(pred, ytest)
    cv$minsplit[i, ] <- c(dev, devmod, mse, misclass, auc1, minsplit_prune_val_best)
}
colnames(cv$cp) <- c("dev", "devmod", "mse", "misclass", "auc", "cp_best")
colnames(cv$dev) <- c("dev", "devmod", "mse", "misclass", "auc", "dev_best")
colnames(cv$minsplit) <- c("dev", "devmod", "mse", "misclass", "auc", "minsplit_best")
```

```
    cv
}

#### RPART cross validation tests

# set.seed(4938868) ## all variables rpartcv_h11 <- rpartCV(x1, h11,
# folds_h11) rpartcv_h04 <- rpartCV(x104, h04, folds_h04) rpartcv_h0411 <-
# rpartCV(x104, h0411, folds_h0411) rpartcv_d11 <- rpartCV(x1, d11,
# folds_d11) rpartcv_d04 <- rpartCV(x104, d04, folds_d04) rpartcv_d0411 <-
# rpartCV(x104, d0411, folds_d0411) ## framington variables rpartcv_h11fram
# <- rpartCV(x1[,framvar11], h11, folds_h11) rpartcv_h04fram <-
# rpartCV(x104[,framvar04], h04, folds_h04) rpartcv_h0411fram <-
# rpartCV(x104[,framvar04], h0411, folds_h0411) set.seed(1748405)
# rpartcv_d11fram <- rpartCV(x1[,framvar11], d11, folds_d11) rpartcv_d04fram
# <- rpartCV(x104[,framvar04], d04, folds_d04) rpartcv_d0411fram <-
# rpartCV(x104[,framvar04], d0411, folds_d0411) ## top 1-10 variables
# rpartcv_h11r <- lapply(1:10, function(num_var) { var <-
# as.character(summary(gbm_h11b)[1:num_var,1]) rpartCV(x[,var,drop=F], h11,
# folds_h11) }) rpartcv_h04r <- lapply(1:10, function(num_var) { var <-
# as.character(summary(gbm_h04b)[1:num_var,1]) rpartCV(x[,var,drop=F], h04,
# folds_h04) }) rpartcv_h0411r <- lapply(1:10, function(num_var) { var <-
# as.character(summary(gbm_h0411b)[1:num_var,1]) rpartCV(x[,var,drop=F],
# h0411, folds_h0411) }) rpartcv_d11r <- lapply(1:10, function(num_var) {
# var <- as.character(summary(gbm_d11b)[1:num_var,1])
# rpartCV(x[,var,drop=F], d11, folds_d11) }) rpartcv_d04r <- lapply(1:10,
# function(num_var) { var <- as.character(summary(gbm_d04b)[1:num_var,1])
# rpartCV(x[,var,drop=F], d04, folds_d04) }) rpartcv_d0411r <- lapply(1:10,
# function(num_var) { var <- as.character(summary(gbm_d0411b)[1:num_var,1])
# rpartCV(x[,var,drop=F], d0411, folds_d0411) }) save(rpartcv_h11,
# rpartcv_h04, rpartcv_h0411, rpartcv_d11, rpartcv_d04, rpartcv_d0411,
# rpartcv_h11fram, rpartcv_h04fram, rpartcv_h0411fram, rpartcv_d11fram,
# rpartcv_d04fram, rpartcv_d0411fram, rpartcv_h11r, rpartcv_h04r,
# rpartcv_h0411r, rpartcv_d11r, rpartcv_d04r, rpartcv_d0411r,
# file='/mnt/DATA/Main/Spring 2015/998/Proj2/models/rpartcv_all.rda')
load("/mnt/DATA/Main/Spring 2015/998/Proj2/models/rpartcv_all.rda")


#### plot rpart cross validation results

getMean <- function(x) mean(x[, "auc"])
getMin <- function(x) min(x[, "auc"])
getMax <- function(x) max(x[, "auc"])
rpartcv_cp <- data.frame(resp = factor(rep(c("HAC 2011", "HAC 2004", "HAC 2004-2011",
    "doc 2011", "doc 2004", "doc 2004-2011"), each = 12), levels = c("HAC 2011",
    "HAC 2004", "HAC 2004-2011", "doc 2011", "doc 2004", "doc 2004-2011")),
    Variables = factor(rep(c(1:10, 304, "Fram"), 6), levels = c(1:10, 304, "Fram")),
    AUC = c(sapply(rpartcv_h11r, function(x) getMean(x$cp)), getMean(rpartcv_h11$cp),
        getMean(rpartcv_h11fram$cp), sapply(rpartcv_h04r, function(x) getMean(x$cp)),
        getMean(rpartcv_h04$cp), getMean(rpartcv_h04fram$cp), sapply(rpartcv_h0411r,
```

```
            function(x) getMean(x$cp)), getMean(rpartcv_h0411$cp), getMean(rpartcv_h0411fram$cp),
        sapply(rpartcv_d11r, function(x) getMean(x$cp)), getMean(rpartcv_d11$cp),
        getMean(rpartcv_d11fram$cp), sapply(rpartcv_d04r, function(x) getMean(x$cp)),
        getMean(rpartcv_d04$cp), getMean(rpartcv_d04fram$cp), sapply(rpartcv_d0411r,
            function(x) getMean(x$cp)), getMean(rpartcv_d0411$cp), getMean(rpartcv_d0411fram$cp))
    min = c(sapply(rpartcv_h11r, function(x) getMin(x$cp)), getMin(rpartcv_h11$cp),
        getMin(rpartcv_h11fram$cp), sapply(rpartcv_h04r, function(x) getMin(x$cp)),
        getMin(rpartcv_h04$cp), getMin(rpartcv_h04fram$cp), sapply(rpartcv_h0411r,
            function(x) getMin(x$cp)), getMin(rpartcv_h0411$cp), getMin(rpartcv_h0411fram$cp),
        sapply(rpartcv_d11r, function(x) getMin(x$cp)), getMin(rpartcv_d11$cp),
        getMin(rpartcv_d11fram$cp), sapply(rpartcv_d04r, function(x) getMin(x$cp)),
        getMin(rpartcv_d04$cp), getMin(rpartcv_d04fram$cp), sapply(rpartcv_d0411r,
            function(x) getMin(x$cp)), getMin(rpartcv_d0411$cp), getMin(rpartcv_d0411fram$cp)),
    max = c(sapply(rpartcv_h11r, function(x) getMax(x$cp)), getMax(rpartcv_h11$cp),
        getMax(rpartcv_h11fram$cp), sapply(rpartcv_h04r, function(x) getMax(x$cp)),
        getMax(rpartcv_h04$cp), getMax(rpartcv_h04fram$cp), sapply(rpartcv_h0411r,
            function(x) getMax(x$cp)), getMax(rpartcv_h0411$cp), getMax(rpartcv_h0411fram$cp),
        sapply(rpartcv_d11r, function(x) getMax(x$cp)), getMax(rpartcv_d11$cp),
        getMax(rpartcv_d11fram$cp), sapply(rpartcv_d04r, function(x) getMax(x$cp)),
        getMax(rpartcv_d04$cp), getMax(rpartcv_d04fram$cp), sapply(rpartcv_d0411r,
            function(x) getMax(x$cp)), getMax(rpartcv_d0411$cp), getMax(rpartcv_d0411fram$cp)))

rpartcv_dev <- data.frame(resp = factor(rep(c("HAC 2011", "HAC 2004", "HAC 2004-2011",
    "doc 2011", "doc 2004", "doc 2004-2011"), each = 12), levels = c("HAC 2011",
    "HAC 2004", "HAC 2004-2011", "doc 2011", "doc 2004", "doc 2004-2011")),
    Variables = factor(rep(c(1:10, 304, "Fram"), 6), levels = c(1:10, 304, "Fram")),
    AUC = c(sapply(rpartcv_h11r, function(x) getMean(x$dev)), getMean(rpartcv_h11$dev),
        getMean(rpartcv_h11fram$dev), sapply(rpartcv_h04r, function(x) getMean(x$dev)),
        getMean(rpartcv_h04$dev), getMean(rpartcv_h04fram$dev), sapply(rpartcv_h0411r,
            function(x) getMean(x$dev)), getMean(rpartcv_h0411$dev), getMean(rpartcv_h0411fram$de
        sapply(rpartcv_d11r, function(x) getMean(x$dev)), getMean(rpartcv_d11$dev),
        getMean(rpartcv_d11fram$dev), sapply(rpartcv_d04r, function(x) getMean(x$dev)),
        getMean(rpartcv_d04$dev), getMean(rpartcv_d04fram$dev), sapply(rpartcv_d0411r,
            function(x) getMean(x$dev)), getMean(rpartcv_d0411$dev), getMean(rpartcv_d0411fram$de
    min = c(sapply(rpartcv_h11r, function(x) getMin(x$dev)), getMin(rpartcv_h11$dev),
        getMin(rpartcv_h11fram$dev), sapply(rpartcv_h04r, function(x) getMin(x$dev)),
        getMin(rpartcv_h04$dev), getMin(rpartcv_h04fram$dev), sapply(rpartcv_h0411r,
            function(x) getMin(x$dev)), getMin(rpartcv_h0411$dev), getMin(rpartcv_h0411fram$dev),
        sapply(rpartcv_d11r, function(x) getMin(x$dev)), getMin(rpartcv_d11$dev),
        getMin(rpartcv_d11fram$dev), sapply(rpartcv_d04r, function(x) getMin(x$dev)),
        getMin(rpartcv_d04$dev), getMin(rpartcv_d04fram$dev), sapply(rpartcv_d0411r,
            function(x) getMin(x$dev)), getMin(rpartcv_d0411$dev), getMin(rpartcv_d0411fram$dev))
    max = c(sapply(rpartcv_h11r, function(x) getMax(x$dev)), getMax(rpartcv_h11$dev),
        getMax(rpartcv_h11fram$dev), sapply(rpartcv_h04r, function(x) getMax(x$dev)),
        getMax(rpartcv_h04$dev), getMax(rpartcv_h04fram$dev), sapply(rpartcv_h0411r,
            function(x) getMax(x$dev)), getMax(rpartcv_h0411$dev), getMax(rpartcv_h0411fram$dev),
        sapply(rpartcv_d11r, function(x) getMax(x$dev)), getMax(rpartcv_d11$dev),
        getMax(rpartcv_d11fram$dev), sapply(rpartcv_d04r, function(x) getMax(x$dev)),
        getMax(rpartcv_d04$dev), getMax(rpartcv_d04fram$dev), sapply(rpartcv_d0411r,
```

```
            function(x) getMax(x$dev)), getMax(rpartcv_d0411$dev), getMax(rpartcv_d0411fram$dev))

rpartcv_minsplit <- data.frame(resp = factor(rep(c("HAC 2011", "HAC 2004", "HAC 2004-2011",
    "doc 2011", "doc 2004", "doc 2004-2011"), each = 12), levels = c("HAC 2011",
    "HAC 2004", "HAC 2004-2011", "doc 2011", "doc 2004", "doc 2004-2011")),
    Variables = factor(rep(c(1:10, 304, "Fram"), 6), levels = c(1:10, 304, "Fram")),
    AUC = c(sapply(rpartcv_h11r, function(x) getMean(x$minsplit)), getMean(rpartcv_h11$minsplit),
        getMean(rpartcv_h11fram$minsplit), sapply(rpartcv_h04r, function(x) getMean(x$minsplit)),
        getMean(rpartcv_h04$minsplit), getMean(rpartcv_h04fram$minsplit), sapply(rpartcv_h0411r,
            function(x) getMean(x$minsplit)), getMean(rpartcv_h0411$minsplit),
        getMean(rpartcv_h0411fram$minsplit), sapply(rpartcv_d11r, function(x) getMean(x$minsplit)),
        getMean(rpartcv_d11$minsplit), getMean(rpartcv_d11fram$minsplit), sapply(rpartcv_d04r,
            function(x) getMean(x$minsplit)), getMean(rpartcv_d04$minsplit),
        getMean(rpartcv_d04fram$minsplit), sapply(rpartcv_d0411r, function(x) getMean(x$minsplit)),
        getMean(rpartcv_d0411$minsplit), getMean(rpartcv_d0411fram$minsplit)),
    min = c(sapply(rpartcv_h11r, function(x) getMin(x$minsplit)), getMin(rpartcv_h11$minsplit),
        getMin(rpartcv_h11fram$minsplit), sapply(rpartcv_h04r, function(x) getMin(x$minsplit)),
        getMin(rpartcv_h04$minsplit), getMin(rpartcv_h04fram$minsplit), sapply(rpartcv_h0411r,
            function(x) getMin(x$minsplit)), getMin(rpartcv_h0411$minsplit),
        getMin(rpartcv_h0411fram$minsplit), sapply(rpartcv_d11r, function(x) getMin(x$minsplit)),
        getMin(rpartcv_d11$minsplit), getMin(rpartcv_d11fram$minsplit), sapply(rpartcv_d04r,
            function(x) getMin(x$minsplit)), getMin(rpartcv_d04$minsplit), getMin(rpartcv_d04fram
        sapply(rpartcv_d0411r, function(x) getMin(x$minsplit)), getMin(rpartcv_d0411$minsplit),
        getMin(rpartcv_d0411fram$minsplit)), max = c(sapply(rpartcv_h11r, function(x) getMax(x$mi
        getMax(rpartcv_h11$minsplit), getMax(rpartcv_h11fram$minsplit), sapply(rpartcv_h04r,
            function(x) getMax(x$minsplit)), getMax(rpartcv_h04$minsplit), getMax(rpartcv_h04fram
        sapply(rpartcv_h0411r, function(x) getMax(x$minsplit)), getMax(rpartcv_h0411$minsplit),
        getMax(rpartcv_h0411fram$minsplit), sapply(rpartcv_d11r, function(x) getMax(x$minsplit)),
        getMax(rpartcv_d11$minsplit), getMax(rpartcv_d11fram$minsplit), sapply(rpartcv_d04r,
            function(x) getMax(x$minsplit)), getMax(rpartcv_d04$minsplit), getMax(rpartcv_d04fram
        sapply(rpartcv_d0411r, function(x) getMax(x$minsplit)), getMax(rpartcv_d0411$minsplit),
        getMax(rpartcv_d0411fram$minsplit)))

# pdf('/mnt/DATA/Main/Spring 2015/998/Proj2/plots/rpartcv_cp.pdf')
# ggplot(rpartcv_cp, aes(y=AUC, x=Variables)) + geom_point(size=3) +
# geom_errorbar(aes(y=AUC, ymin=min, ymax=max)) + facet_wrap(~resp) +
# theme(strip.text=element_text(size=18), axis.text=element_text(angle=90,
# color='black', size=12), axis.title=element_text(size=12)) + ylim(.4,.85)
# dev.off() pdf('/mnt/DATA/Main/Spring
# 2015/998/Proj2/plots/rpartcv_dev.pdf') ggplot(rpartcv_dev, aes(y=AUC,
# x=Variables)) + geom_point(size=3) + geom_errorbar(aes(y=AUC, ymin=min,
# ymax=max)) + facet_wrap(~resp) + theme(strip.text=element_text(size=18),
# axis.text=element_text(angle=90, color='black', size=12),
# axis.title=element_text(size=12)) + ylim(.4,.85) dev.off()
# pdf('/mnt/DATA/Main/Spring 2015/998/Proj2/plots/rpartcv_minsplit.pdf')
# ggplot(rpartcv_minsplit, aes(y=AUC, x=Variables)) + geom_point(size=3) +
# geom_errorbar(aes(y=AUC, ymin=min, ymax=max)) + facet_wrap(~resp) +
# theme(strip.text=element_text(size=18), axis.text=element_text(angle=90,
# color='black', size=12), axis.title=element_text(size=12)) + ylim(.4,.85)
```

```
# dev.off()

## fit rpart with framington variables using deviance pruning
## set.seed(142111) rpartprune_h11fram <- rpartPruneCV(x[,framvar11], h11,
## folds_h11) rpartprune_h04fram <- rpartPruneCV(x[,framvar04], h04,
## folds_h04) rpartprune_h0411fram <- rpartPruneCV(x[,framvar04], h0411,
## folds_h0411) rpartprune_d11fram <- rpartPruneCV(x[,framvar11], d11,
## folds_d11) rpartprune_d04fram <- rpartPruneCV(x[,framvar04], d04,
## folds_d04) rpartprune_d0411fram <- rpartPruneCV(x[,framvar04], d0411,
## folds_d0411) rpart_h11fram <- rpart(h11~., data=x[,framvar11],
## method='class', control=rpart.control(maxsurrogate=100, cp=-1))
## rpart_h04fram <- rpart(h04~., data=x[,framvar04], method='class',
## control=rpart.control(maxsurrogate=100, cp=-1)) rpart_h0411fram <-
## rpart(h11~., data=x[,framvar04], method='class',
## control=rpart.control(maxsurrogate=100, cp=-1)) rpart_d11fram <-
## rpart(d11~., data=x[,framvar11], method='class',
## control=rpart.control(maxsurrogate=100, cp=-1)) rpart_d04fram <-
## rpart(h11~., data=x[,framvar04], method='class',
## control=rpart.control(maxsurrogate=100, cp=-1)) rpart_d0411fram <-
## rpart(h11~., data=x[,framvar04], method='class',
## control=rpart.control(maxsurrogate=100, cp=-1)) save(rpartprune_h11fram,
## rpartprune_h04fram, rpartprune_h0411fram, rpartprune_d11fram,
## rpartprune_d04fram, rpartprune_d0411fram, rpart_h11fram, rpart_h04fram,
## rpart_h0411fram, rpart_d11fram, rpart_d04fram, rpart_d0411fram,
## file='/mnt/DATA/Main/Spring 2015/998/Proj2/models/rpart_fram.rda')
load("/mnt/DATA/Main/Spring 2015/998/Proj2/models/rpart_fram.rda")

dev_h11fram <- getPruneValBest(rpartprune_h11fram, "auc", "dev")
dev_h04fram <- getPruneValBest(rpartprune_h04fram, "auc", "dev")
dev_h0411fram <- getPruneValBest(rpartprune_h0411fram, "auc", "dev")
dev_d11fram <- getPruneValBest(rpartprune_d11fram, "auc", "dev")
dev_d04fram <- getPruneValBest(rpartprune_d04fram, "auc", "dev")
dev_d0411fram <- getPruneValBest(rpartprune_d0411fram, "auc", "dev")
minsplit_h11fram <- getPruneValBest(rpartprune_h11fram, "auc", "minsplit")
minsplit_h04fram <- getPruneValBest(rpartprune_h04fram, "auc", "minsplit")
minsplit_h0411fram <- getPruneValBest(rpartprune_h0411fram, "auc", "minsplit")
minsplit_d11fram <- getPruneValBest(rpartprune_d11fram, "auc", "minsplit")
minsplit_d04fram <- getPruneValBest(rpartprune_d04fram, "auc", "minsplit")
minsplit_d0411fram <- getPruneValBest(rpartprune_d0411fram, "auc", "minsplit")

plotdir <- "/mnt/DATA/Main/Spring 2015/998/Proj2/plots/"

pdf(paste0(plotdir, "dev_h11fram.pdf"))
prp(myprune.rpart(rpart_h11fram, "dev", dev_h11fram), type = 4, extra = 7)
dev.off()
pdf(paste0(plotdir, "dev_h04fram.pdf"))
prp(myprune.rpart(rpart_h04fram, "dev", dev_h04fram), type = 4, extra = 7)
dev.off()
pdf(paste0(plotdir, "dev_h0411fram.pdf"))
```

```
prp(myprune.rpart(rpart_h0411fram, "dev", dev_h0411fram), type = 4, extra = 7)
dev.off()
pdf(paste0(plotdir, "dev_d11fram.pdf"))
prp(myprune.rpart(rpart_d11fram, "dev", dev_d11fram), type = 4, extra = 7)
dev.off()
pdf(paste0(plotdir, "dev_d04fram.pdf"))
prp(myprune.rpart(rpart_d04fram, "dev", dev_d04fram), type = 4, extra = 7)
dev.off()
pdf(paste0(plotdir, "dev_d0411fram.pdf"))
prp(myprune.rpart(rpart_d0411fram, "dev", dev_d0411fram), type = 4, extra = 7)
dev.off()

pdf(paste0(plotdir, "minsplit_h11fram.pdf"))
prp(myprune.rpart(rpart_h11fram, "minsplit", minsplit_h11fram), type = 4, extra = 7)
dev.off()
pdf(paste0(plotdir, "minsplit_h04fram.pdf"))
prp(myprune.rpart(rpart_h04fram, "minsplit", minsplit_h04fram), type = 4, extra = 7)
dev.off()
pdf(paste0(plotdir, "minsplit_h0411fram.pdf"))
prp(myprune.rpart(rpart_h0411fram, "minsplit", minsplit_h0411fram), type = 4,
    extra = 7)
dev.off()
pdf(paste0(plotdir, "minsplit_d11fram.pdf"))
prp(myprune.rpart(rpart_d11fram, "minsplit", minsplit_d11fram), type = 4, extra = 7)
dev.off()
pdf(paste0(plotdir, "minsplit_d04fram.pdf"))
prp(myprune.rpart(rpart_d04fram, "minsplit", minsplit_d04fram), type = 4, extra = 7)
dev.off()
pdf(paste0(plotdir, "minsplit_d0411fram.pdf"))
prp(myprune.rpart(rpart_d0411fram, "minsplit", minsplit_d0411fram), type = 4,
    extra = 7)
dev.off()
```

## Appendix D: Additional note

Throughout this report, we tried to adhere to the style suggested by Leek.[7] We used the R statistical environment for all calculations[8]. Mike Wurm performed the tree-based analysis, and the code for that part is nearly identical to his. The R packages gbm,[9] pROC[10] and rpart[11] played central roles in our analyses.

## References


D'Agostino, Ralph B, Ramachandran S Vasan, Michael J Pencina, Philip A Wolf, Mark Cobain, Joseph M Massaro, and William B Kannel. "General Cardiovascular Risk Profile for Use in Primary Care the Framingham Heart Study." *Circulation* 117, no. 6 (2008): 743–53.

Leek, Jeff. *The Elements of Data Analytic Style*. Leanpub, 2015Available at https://leanpub.com/datastyle.

Loh, Wei-Yin. "Classification and Regression Trees." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1, no. 1 (2011): 14–23.

Mozaffarian, Dariush, Emelia J Benjamin, Alan S Go, Donna K Arnett, Michael J Blaha, Mary Cushman, Sarah de Ferranti, et al. "Executive Summary: Heart Disease and Stroke Statistics—2015 Update a Report from the American Heart Association." *Circulation* 131, no. 4 (2015): 434–41.

others, Greg Ridgeway with contributions from. *Gbm: Generalized Boosted Regression Models*, 2015. http://CRAN.R-project.org/package=gbmR package version 2.1.1.

R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2015. http://www.R-project.org/.

Robin, Xavier, Natacha Turck, Alexandre Hainard, Natalia Tiberti, Frédérique Lisacek, Jean-Charles Sanchez, and Markus Müller. "PROC: An Open-Source Package for R and S+ to Analyze and Compare ROC Curves." *BMC Bioinformatics* 12 (2011): 77.

Therneau, Terry, Beth Atkinson, and Brian Ripley. *Rpart: Recursive Partitioning and Regression Trees*, 2015. http://CRAN.R-project.org/package=rpartR package version 4.1-9.

"Wisconsin Longitudinal Study". http://www.ssc.wisc.edu/wlsresearch/, 2015Accessed on 21-March-2015.


[7] *The Elements of Data Analytic Style.*
[8] R Core Team, *R.*
[9] Others, *Gbm.*
[10] Robin et al., "PROC."
[11] Therneau, Atkinson, and Ripley, *Rpart.*