STA135 - HW05 - Boosting and Bagging

Hugh Crockford

March 14, 2013

1 Random Forest

Random forest was grown on training data and error examined, see below for confusion matrix.

	0	1	2	3	4	5	6	7	8	9
0	96.02	0.00	0.29	0.14	0.15	0.63	0.68	0.11	0.79	0.11
1	0.00	97.62	0.64	0.39	0.22	0.20	0.25	0.21	0.34	0.11
2	0.47	0.32	93.12	0.88	0.83	0.39	0.75	1.09	1.17	0.49
3	0.43	0.26	2.00	91.84	0.25	3.06	0.25	0.91	2.04	1.10
4	0.18	0.00	0.46	0.14	92.82	0.31	0.79	0.39	0.72	2.93
5	1.04	0.23	0.32	2.52	0.29	91.46	1.25	0.25	1.25	0.92
6	0.90	0.16	0.32	0.11	0.40	1.06	94.96	0.00	0.79	0.07
7	0.18	0.55	1.43	0.35	1.09	0.12	0.04	95.27	0.45	2.12
8	0.36	0.64	0.89	2.35	1.05	2.08	0.93	0.21	91.38	1.80
9	0.43	0.23	0.53	1.30	2.90	0.71	0.11	1.58	1.06	90.35

Table 1: confusion matrix for Random Forest, ntree = 30

When the confusion for each class is compared to Out of Bag error rate, the following graph is generated showing oob error is greater than the error for some digits, and less for others. The mean of clas error is 6.5%, which is very close to OOB rate of 6.44%

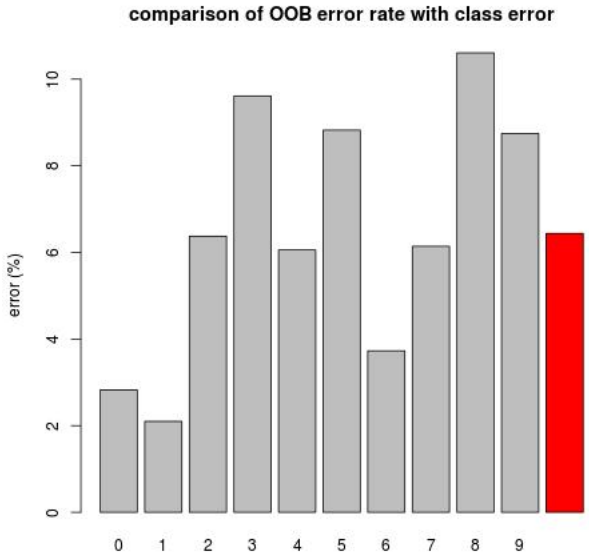

Figure 1: Summary of Accuracy for each digit vs overall OOB error rate(red)

2 Importance

The importance of each pixel sorted by importance can be seen in following plot: The plot shows the first 50 most important pixels are responsible for 32% of the change in gine, and the first 100 were 52%

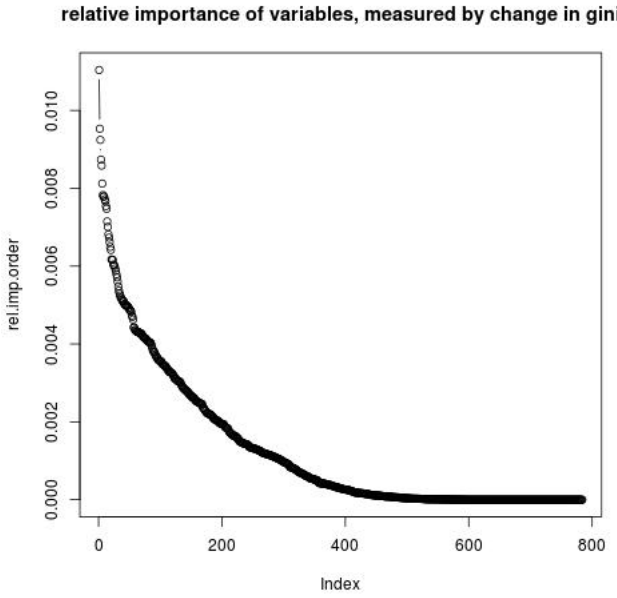

Figure 2: Importance of each pixel

A heatmap showing pixels colored by importance can be seen below. Although it is fairly homogenous some interesting things can be seen:

1. most of the inmportant pixels are in top part of the image.

2. specifically the top 4 are arranged around what would be considered the upper circle of digits 3,8,9, but would also be an important part of digits 2,3,5,6, and 7. This makes intuitive sense that the intensity of these pixels would predict the digit well.

3. Likewise there is a band down left and bottom of image, corresponding to parts of 1,2,4,6,8 and 0.

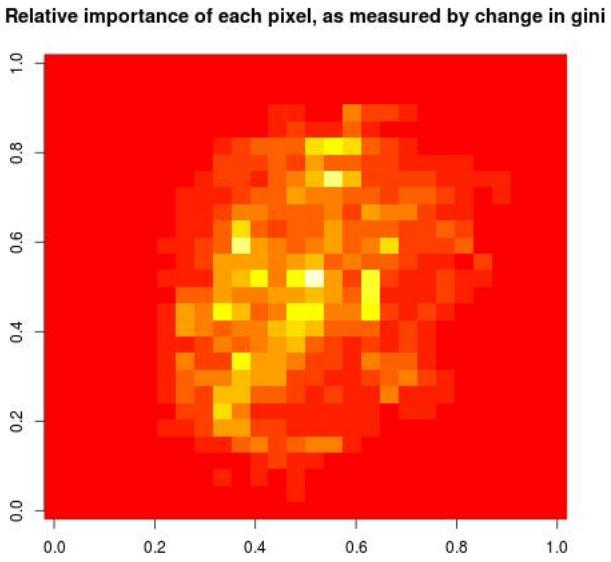

Figure 3: Importance of each pixel shown on heatmap.

2.1 Timing - speeding up RF

I was interested in the effect of only using these important pixels and growing same, or more trees on the accuracy of prediction.

OOB rates on first 50 most important pixels with 30 trees was 22% in only 7 seconds, a considerable jump from the 6.4% with all pixels, although this initial model took 2202 seconds. Using the first 100 most important pixels with 30 trees the OOB was 15% and it took 12 seconds

When I increased number of trees to default 500, the first 50 yielded 18.14% OOB in 116 seconds, while the first 100 yielded 11% OOB and took 205 seconds.

It seems from this brief investigation into time vs accuracy tradeoff that using a subset of initial inputs with few trees produces a fast result, that retains much of the accuracy of the full model (It should be noted, however, that for this dataset QDA performed just as well as RF w 30 trees, in a fraction of the time.

3 Testdata

Using the test data, the accuracy was similar to the crossvalidated training set.

	0	1	2	3	4	5	6	7	8	9
0	97.19	0.00	0.22	0.07	0.15	0.24	0.50	0.07	0.30	0.14
1	0.00	98.17	0.58	0.28	0.29	0.00	0.29	0.20	0.22	0.14
2	0.29	0.25	95.48	0.77	0.52	0.08	0.93	1.01	1.27	0.21
3	0.07	0.25	1.38	94.72	0.07	3.12	0.21	0.61	0.97	1.07
4	0.22	0.19	0.22	0.00	95.51	0.16	0.36	0.13	0.30	2.00
5	0.43	0.06	0.07	1.27	0.15	93.67	0.93	0.13	0.97	0.64
6	0.72	0.06	0.07	0.00	0.22	1.04	95.92	0.00	0.52	0.00
7	0.29	0.19	1.24	0.14	0.44	0.08	0.00	96.70	0.52	1.79
8	0.29	0.44	0.58	1.41	0.74	1.12	0.79	0.13	94.18	1.43
9	0.50	0.38	0.15	1.34	1.91	0.48	0.07	1.01	0.75	92.57

Table 2: confusion matrix for Random Forest, test data, full model

To compare the accuracy on test vs training set, I took a ratio of test/-train, figuring if they were similar al values should be around 1.

	0	1	2	3	4	5	6	7	8	9
0	96.02	0.00	0.29	0.14	0.15	0.63	0.68	0.11	0.79	0.11
1	0.00	97.62	0.64	0.39	0.22	0.20	0.25	0.21	0.34	0.11
2	0.47	0.32	93.12	0.88	0.83	0.39	0.75	1.09	1.17	0.49
3	0.43	0.26	2.00	91.84	0.25	3.06	0.25	0.91	2.04	1.10
4	0.18	0.00	0.46	0.14	92.82	0.31	0.79	0.39	0.72	2.93
5	1.04	0.23	0.32	2.52	0.29	91.46	1.25	0.25	1.25	0.92
6	0.90	0.16	0.32	0.11	0.40	1.06	94.96	0.00	0.79	0.07
7	0.18	0.55	1.43	0.35	1.09	0.12	0.04	95.27	0.45	2.12
8	0.36	0.64	0.89	2.35	1.05	2.08	0.93	0.21	91.38	1.80
9	0.43	0.23	0.53	1.30	2.90	0.71	0.11	1.58	1.06	90.35

Table 3: Ratio of confusion matrix for test vs train data

This was difficult to interpret so I represented as an image, with yellow representing values around 1, and red those that are well below 1.

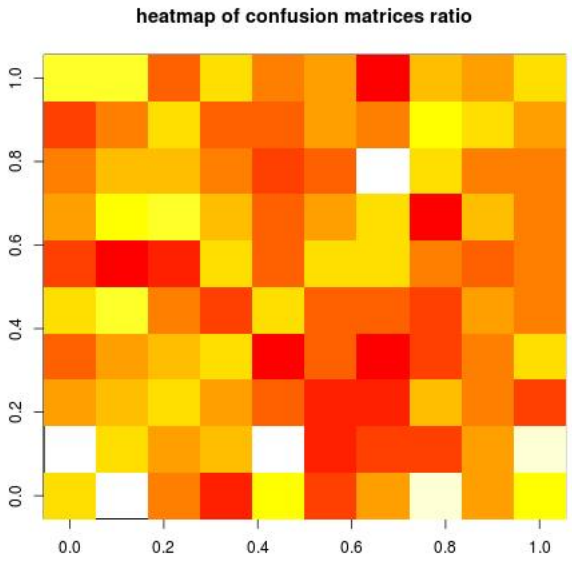

Figure 4: Heatmap of confusion matricies ratio

From this image it can be seen the training set was more accurate on 5-8 vs 2-4, and the test set struggled with 4-6 vs 8-10.

4 Boosting

5 CODE

```
# STA 135 H05
# 20130312 HCrockford

library(MASS)
library(rpart)
library(xtable)
library(randomForest)
load("../H04/digitsKnownTest.rda")
load("../H04/digitsTrain.rda")

######
## Random Forest
######

fullrf = randomForest(as.factor(label) ~ ., data =
    sampleTrain, ntree=30)        # choose 30 - from
    previous hw gains drop off after this.
conf = rf$confusion[,11]*100
oob = 6.44
names(oob) = "OOB err"

xtab = xtable(round(prop.table(rf$confusion[,-11],
    margin=2)*100,2), caption = "confusion matrix for
    Random Forest, ntree = 30")
print(xtab,file = 'confRF.tex')

jpeg('err.jpg')
barplot(c(conf,oob),main="comparison of OOB error rate
    with class error",ylab = "error (%)",col = c(rep("
    grey",10),"red"))
dev.off()

imp = rf$importance
impp = transform(vnom = row.names(imp),imp)
impp.order = impp[order(imp,decreasing = TRUE),]
```

```
imp.order = imp[order(imp,decreasing = TRUE),]
rel.imp.order = imp.order/sum(imp.order)

jpeg('relplot.jpg')
plot(rel.imp.order,type = "b",main = "relative
   importance of variables, measured by change in gini
   ")
dev.off()

img = matrix(imp,nrow=28)
image(img,nrow=2)
rotmat = diag(rep(1, times=28))[28:1,]
newimg = img%*%rotmat

jpeg('imp.jpg')
image(newimg,main = "Relative importance of each pixel,
    as measured by change in gini")
dev.off()

small50 = sampleTrain[,impp.order[1:50,2]] # get 50
   most important
rf5030 = randomForest(as.factor(sampleTrain$label) ~ .,
    data = small50, ntree=30)       # choose 30 - from
   previous hw gains drop off after this.
rf50 = randomForest(as.factor(sampleTrain$label) ~ .,
   data = small50)   # choose 30 - from previous hw
   gains drop off after this.

small100 = sampleTrain[,impp.order[1:100,2]] # get 50
   most important
rf10030 = randomForest(as.factor(sampleTrain$label) ~
   ., data = small100, ntree=30)     # choose 30 - from
    previous hw gains drop off after this.
rf100 = randomForest(as.factor(sampleTrain$label) ~ .,
   data = small100) # choose 30 - from previous hw
   gains drop off after this.

system.time(randomForest(as.factor(sampleTrain$label) ~
```

```
    . data = small50, ntree=30)

res.fullrf = predict(fullrf,test[,-1],class = TRUE)
tab = table(true = test[,1],pred = res.fullrf)

xtab = xtable(round(prop.table(tab,margin=2)*100,2),
   caption = "confusion matrix for Random Forest, test
   data, full model")
print(xtab,file = 'conftestRF.tex')
```

```
############################
## BOOST try
############################

train = sampleTrain
train.e = eigen(cov(train[,-1]))
train.pca = as.matrix(train[,-1]) %*% train.e$vectors
   [,1:20]    # transform data to top 20 eigens
dat = cbind(label = as.factor(train$label), data.frame(
```

```
    train.pca))

train.lda = lda(as.factor(label) ~ ., data = dat, CV =
    FALSE) # LDA doesnent supoport weights!!
train.pred = predict(train.lda, newdata = data.frame(
    dat[,-1]))


tree4 = rpart(as.factor(label) ~ . , data = train,
    maxdepth = 4)
test.pred = predict(tree3, newdata = test[,-1],type = "
    class")
wrong = test$label != test.pred
sum(wrong)/length(wrong)          # getting 36% -
    dooable.

tree = list()
train.pred = list()
alpha = list()
epsilon = list()
wt =  rep(1/length(train$label),length(train$label)) #
    init wt.
i=1

wt =  rep(1/length(train$label),length(train$label)) #
    init wt.
tree = list()
got  = for(i in 1:10){
        tree[[i]] = rpart(as.factor(label) ~ . , data =
            train, weights = wt,maxdepth = 4)
        train.pred = predict(tree[[i]], newdata = train
            [,-1],type = "class")
        wrong = train$label != train.pred
        err = sum(wrong)/length(wrong)
        epsilon = sum(wt[wrong])/sum(wt)          # calc
            error
        alpha = log((1-epsilon)/epsilon,base = exp(1))
            # calc alpha
```

```
        wt[wrong] = wt[wrong] * exp(alpha)      # update
            weights
        print(c(i,alpha,epsilon,err))
 }

table(wt)

tree5 = rpart(as.factor(label) ~ . , data = train,
   weights = wt,maxdepth = 4)
train.pred = predict(tree3, newdata = train[,-1],type =
    "class")
wrong = train$label != train.pred
epsilon = sum(wt[wrong])/sum(wt)          # calc error
alpha = log((1-epsilon)/epsilon,base = exp(1)) # calc
   alpha
wt[wrong] = wt[wrong] * exp(alpha)      # update weights
return(c(i,alpha,epsilon))

###################################
## RF try
##################

# need to determine best split out of all variables
## split that maximises difference in groups.

# play data

t = data.frame(lab = rep(c(1,0),each = 10),x = c(sample
   (6:10,10,replace = TRUE),sample(1:8,10,replace =
   TRUE))) #  best cut should be 7.
mean(t$x)                                 #
pred = 1* (t$x > mean(t$x))               # starting point
    - work from mean
isBetter = TRUE
while(isBetter == TRUE){

        right2 = sum(t$lab == pred)/length(t$lab)
        if(right2>right){isBetter == TRUE}
```

```
        print right2
}

bestcut = function(dat,lab = minitrain$label) { #
   return best cut for binary - how do multiple groups?
      scor = sapply(unique(dat), function(i){
             sum((1*dat>i) == unique(lab)*c(TRUE,FALSE)
                )/length(lab)
})
        c(unique(dat)[which.max(scor)],max(scor))
}


hick = subset(train,label %in% c(1,8))
minitrain = hick[1:100,]

bestcut(minitrain[,689],minitrain$label)      #
   returning cutoff and purity.

res = apply(minitrain[,-1],2,bestcut)
```