Tanner Schulz

4/13/23

STAT 5650

Homework 5

1. This problem continues the analysis of the *Forensic Glass* data.
    1. a) Apply random forests to the data and obtain the out-of-bag confusion matrix. How well can we classify these data, and where are the major misclassifications? How do your results compare to the classification tree you fitted in Homework #3?

```
library(randomForest)
library(verification)

glasses = read.csv("Glass copy.csv")

# # #
# a #
# # #

glass.rf=randomForest(as.factor(GlassType)~ . ,data=glasses)

glass.rf$confusion
glass.rf.confusion <- table(glasses$GlassType,predict(glass.rf,type="response"))
100*sum(diag(glass.rf.confusion))/sum(glass.rf.confusion)
```

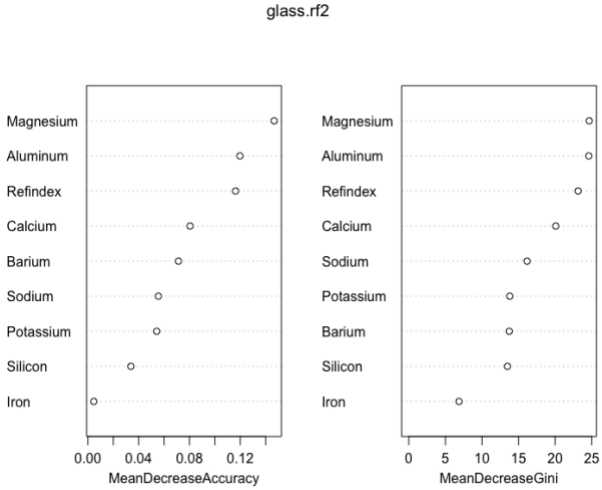
```
> glass.rf=randomForest(as.factor(GlassType)~ . ,data=glasses)
> glass.rf$confusion
   1  2 3  4 5  6 class.error
1 62  7 1  0 0  0   0.1142857
2 10 61 1  2 1  1   0.1973684
3  8  2 7  0 0  0   0.5882353
4  0  2 0 10 0  1   0.2307692
5  0  2 0  0 7  0   0.2222222
6  1  2 0  0 0 26   0.1034483
> glass.rf.confusion <- table(glasses$GlassType,predict(glass.rf,type="response"))
> 100*sum(diag(glass.rf.confusion))/sum(glass.rf.confusion)
[1] 80.84112
```

Looking at the OOB matrix, the major misclassifications seemed to be within GlassType 3. There were 17 GlassType 3s with only 7 correctly classified. 8 of the 10 misclassifications happened within the GlassType 1 column and 2 within the GlassType 2 column. Those seem to be the major misclassifications that lead to GlassType 3 having a class error of 0.5882.

When compared to the classification tree fit in the previous homework the random forest performed much better. While it only somewhat alleviated the problem with GlassType 3 being misclassified majorly, (16 out of 17 were misclassifications in the decision tree). The decision tree still only yielded an accuracy of 65.42%. This simple random forest model yielded an accuracy of 80.84%.

2. b) Use random forests to select a subset of the variables (which may be all the variables!) Refit random forests with only the important variables and obtain the out-of-bag confusion matrix. Did you observe any change in predictive accuracy?

```
# # #
# b #
# # #

glass.rf2=randomForest(as.factor(GlassType)~ . ,importance=TRUE,data=glasses)

varImpPlot(glass.rf2,scale=FALSE)
```



Looking at the variable importance plot, I want to try two different models. One taking the only the top 4 (Magnesium, Aluminum, Refindex, and Calcium) in terms of variable importance and the other looking at everything except the bottom 2 (Silicon and Iron).

Top 4 model:

```
# model using the top 4
glass.rf2.top4=randomForest(as.factor(GlassType)~ Magnesium+Aluminum+Refindex+Calcium ,data=glasses)

glass.rf2.top4$confusion
glass.rf2.confusion=table(glasses$GlassType,predict(glass.rf2.top4,type="response"))
100*sum(diag(glass.rf2.confusion))/sum(glass.rf2.confusion)
```

```
> glass.rf2.top4$confusion
   1  2 3 4 5  6 class.error
1 62  6 2 0 0  0   0.1142857
2  8 58 4 2 3  1   0.2368421
3  8  2 7 0 0  0   0.5882353
4  0  1 0 8 1  3   0.3846154
5  0  2 0 2 4  1   0.5555556
6  1  3 0 1 0 24   0.1724138
> glass.rf2.confusion=table(glasses$GlassType,predict(glass.rf2.top4,type="response"))
> 100*sum(diag(glass.rf2.confusion))/sum(glass.rf2.confusion)
[1] 76.16822
```

This model didn't perform how I thought that it would. It achieved an accuracy of 76.17% and now has major problems with classification in GlassType 5 as well as GlassType 3. I'll scrap this model and move onto the other model I had in mind.

Everything except bottom 2 model:

```
# model using not bottom 2
glass.rf2.top5=randomForest(as.factor(GlassType)~ .-Iron-Silicon,data=glasses)

glass.rf2.top5$confusion
glass.rf2.confusion2=table(glasses$GlassType,predict(glass.rf2.top5,type="response"))
100*sum(diag(glass.rf2.confusion2))/sum(glass.rf2.confusion2)
```

```
> # model using not bottom 2
> glass.rf2.top5=randomForest(as.factor(GlassType)~ .-Iron-Silicon,data=glasses)
> glass.rf2.top5$confusion
   1  2 3  4 5  6 class.error
1 63  5 2  0 0  0   0.1000000
2  8 60 3  3 1  1   0.2105263
3  7  2 8  0 0  0   0.5294118
4  0  1 0 11 0  1   0.1538462
5  0  2 0  0 7  0   0.2222222
6  1  3 0  0 0 25   0.1379310
> glass.rf2.confusion2=table(glasses$GlassType,predict(glass.rf2.top5,type="response"))
> 100*sum(diag(glass.rf2.confusion2))/sum(glass.rf2.confusion2)
[1] 81.30841
```

By getting rid of Iron and Silicon within the random forest it actually attained a higher accuracy than the first model. This model achieved an accuracy of 81.31%. While it still suffers from major misclassifications within GlassType 3, it actually correctly classified one more brining the class error down a bit. Considering this and that there was a positive change in predictive accuracy, I would say that this is a better model.

3. c) Summarize your results for your analyses of the forensic glass data using classification trees and random forests.

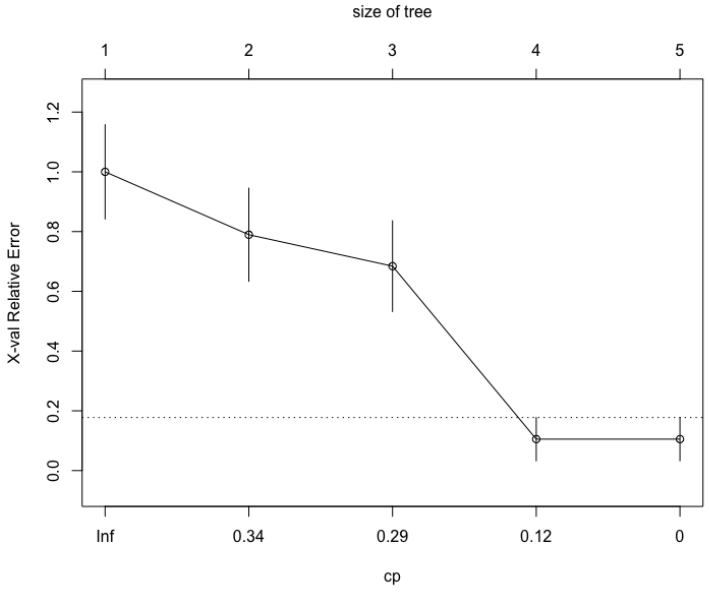
The results were pretty much how I expected going into this analysis. It seems that the random forest models overwhelmingly outperformed the previous decision tree models created in homework 4. The decision tree model found in the previous homework had a predictive accuracy of 65.42%. While the best computed random forest model (bottom 2 variables excluded) had predictive accuracy of 81.31%. The misclassifications were less spread overall and problems with the decision tree model for the most part were alleviated. Given all of this information, I'm curious how boosting would perform on the glass data. If I have time, I'll come back to it.

2. For this question I have uploaded some data on the chemical analyses of samples of orange juice from different countries (**neworange.csv**). The dual purposes of analyzing these data is to determine whether it is possible to determine the country of origin of orange juice based upon this kind of chemical analysis and to identify which variables are important to the classification.

For this analysis I will use both a decision tree as well as a random forest on this data to try and classify which country the oranges come from. First, I will start with a decision tree (this should be essentially a baseline for the random forest as I assume it will perform better).

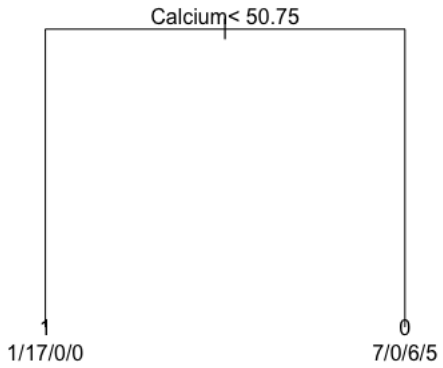
Decision tree model:

```
# # # # #
# # # # #
# # 2 # #
# # # # #
# # # # #

orange = read.csv("neworangelabeled.csv")

# I quickly went through and dummy labeled the countries in excel.
# BEL = 0
# LSP = 1
# TME = 2
# VME = 3

# decision tree model
library(rpart)
# cp plot
orange.rpartfull=rpart(Country~ .-FDA_ID-Unique_ID ,method="class",
                       control=rpart.control(cp=0.0,minsplit=2),data=orange)
plot(orange.rpartfull)
plotcp(orange.rpartfull)
```



First thing to do is to identify the cp needed for the decision tree model. From this we can determine that 0.12 or 0 would be good cp values. I will go with the 0.12 cp value. Additionally for this analysis I will exclude the unique identifiers off the bat as they are essentially just keys and give no insight.

```
# I will use a cp of 0.12 for this decision tree
orange.rpartcp12=rpart(Country~ .
                       ,method="class",data=orange,control=rpart.control(cp=0.12))
plot(orange.rpartcp12,margin=0.1)
text(orange.rpartcp12,use.n=TRUE)
orange.rpartcp12
table(orange$Country,predict(orange.rpartcp12,type="class"))
xvs=rep(c(1:10),length=nrow(orange))
xvs=sample(xvs)
orange.rpartcp12.xval.predprob=rep(0,length(nrow(orange)))
orange.rpartcp12.xval.predclass=rep(0,length(nrow(orange)))
for(i in 1:10){
  train=orange[xvs!=i,]
  test=orange[xvs==i,]
  rp=rpart(Country~ .
           ,method="class",data=train,control=rpart.control(cp=0.12))
  orange.rpartcp12.xval.predprob[xvs==i]=predict(rp,test,type="prob")[,2]
  orange.rpartcp12.xval.predclass[xvs==i]=predict(rp,test,type="class")
}
oranget = table(orange$Country, round(orange.rpartcp12.xval.predclass))
oranget
100*sum(diag(oranget))/sum(oranget)
```




```
> oranget

    1  2
  0 6  2
  1 0 17
  2 6  0
  3 5  0
> 100*sum(diag(oranget))/sum(oranget)
[1] 63.88889
```

The results for this were very underwhelming. It is above the majority classifier though with a prediction accuracy of 63.89%. This decision tree only made one split on Calcium and sorted the data into two buckets. There isn't too much to say about this model though mainly due to it being a baseline that will hopefully be heavily outperformed by a random forest model.

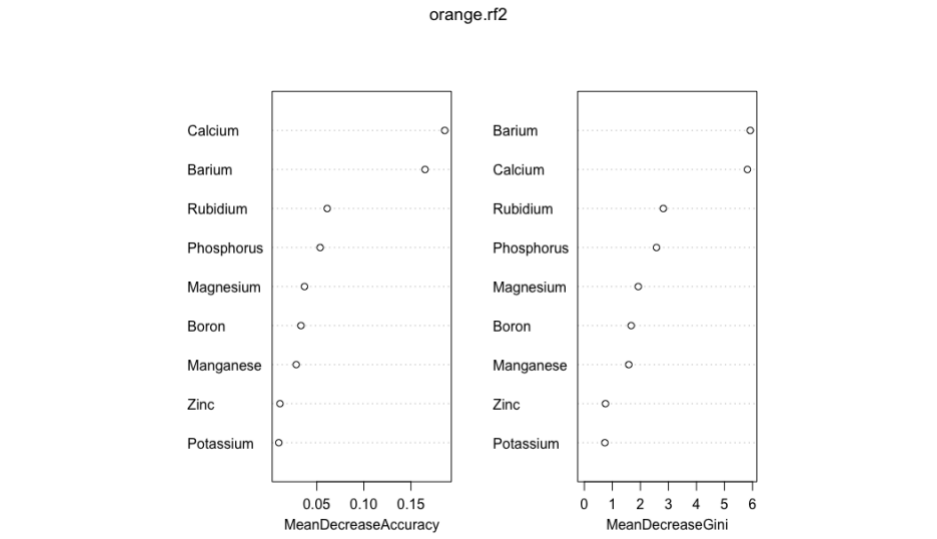
Random Forest model:

```
# random forest

orange.rf=randomForest(as.factor(Country)~ . ,data=orange)

orange.rf$confusion
orange.rf.confusion <- table(orange$Country,predict(orange.rf,type="response"))
100*sum(diag(orange.rf.confusion))/sum(orange.rf.confusion)
```

```
> orange.rf=randomForest(as.factor(Country)~ . ,data=orange)
> orange.rf$confusion
  0  1 2 3 class.error
0 6  2 0 0        0.25
1 0 17 0 0        0.00
2 0  0 6 0        0.00
3 0  0 0 5        0.00
> orange.rf.confusion <- table(orange$Country,predict(orange.rf,type="response"))
> 100*sum(diag(orange.rf.confusion))/sum(orange.rf.confusion)
[1] 94.44444
```

With a simple random forest model, it is able to get an accuracy of 94.44% on the orange data. The only misclassifications were within BEL where two BEL were predicted to be LSP. This is a great model, but let's try and simplify it by cutting out some unimportant variables.

```
# cutting variables
orange.rf2=randomForest(as.factor(Country)~ . ,importance=TRUE,data=orange)

varImpPlot(orange.rf2,scale=FALSE)
```



Like before I will try out two models. The first is going to take out the bottom 3 variables (Manganese, Zinc, and Potassium) while the second will take out the bottom 4 variables (Boron, Manganese, Zinc, and Potassium). If any additional models are needed I will do those as well.

Bottom 3 variables cut:

```
# taking out bottom 3
orange.rf2.bot3=randomForest(as.factor(Country)~ .-Manganese-Zinc-Potassium,data=orange)

orange.rf2.bot3$confusion
orange.rf2.confusion2=table(orange$Country,predict(orange.rf2.bot3,type="response"))
100*sum(diag(orange.rf2.confusion2))/sum(orange.rf2.confusion2)
```

```
> orange.rf2.bot3$confusion
  0  1 2 3 class.error
0 6  2 0 0        0.25
1 0 17 0 0        0.00
2 0  0 6 0        0.00
3 0  0 0 5        0.00
> orange.rf2.confusion2=table(orange$Country,predict(orange.rf2.bot3,type="respons
e"))
> 100*sum(diag(orange.rf2.confusion2))/sum(orange.rf2.confusion2)
[1] 94.44444
```

Taking out the bottom 3 variables resulted in the same model as all of the variables included. Exact same misclassifications and accuracy of 94.44%. Since there is no change with this model taking out 3 variables, I'll move onto taking out more with the bottom 4 variables cut model.

Bottom 4 variables cut:

```
# taking out bottom 4
orange.rf2.bot4=randomForest(as.factor(Country)~ .-Boron-Manganese-Zinc-Potassium,data=orange)

orange.rf2.bot4$confusion
orange.rf2.confusion3=table(orange$Country,predict(orange.rf2.bot4,type="response"))
100*sum(diag(orange.rf2.confusion3))/sum(orange.rf2.confusion3)
```

```
> # taking out bottom 4
> orange.rf2.bot4=randomForest(as.factor(Country)~ .-Boron-Manganese-Zinc-Potassium,
data=orange)
> orange.rf2.bot4$confusion
  0  1 2 3 class.error
0 6  2 0 0        0.25
1 0 17 0 0        0.00
2 0  0 6 0        0.00
3 0  0 0 5        0.00
> orange.rf2.confusion3=table(orange$Country,predict(orange.rf2.bot4,type="respons
e"))
> 100*sum(diag(orange.rf2.confusion3))/sum(orange.rf2.confusion3)
[1] 94.44444
```

Again, this is the same model as before. Same misclassifications, same accuracy of 94.44%. Interesting, since that is the case, I will go further until there is a drop in accuracy or change in misclassifications.

Bottom 8 variables cut:

```
# taking out bottom 7
orange.rf2.bot7=randomForest(as.factor(Country)~ .-Rubidium-Phosphorus-Magnesium-Boron-Manganese-Zinc-Potassium,
                        data=orange)

orange.rf2.bot7$confusion
orange.rf2.confusion4=table(orange$Country,predict(orange.rf2.bot7,type="response"))
100*sum(diag(orange.rf2.confusion4))/sum(orange.rf2.confusion4)
```

```
> # taking out bottom 7
> orange.rf2.bot7=randomForest(as.factor(Country)~ .-Rubidium-Phosphorus-Magnesium-B
oron-Manganese-Zinc-Potassium,
+                              data=orange)
> orange.rf2.bot7$confusion
  0  1 2 3 class.error
0 6  2 0 0        0.25
1 0 17 0 0        0.00
2 0  0 6 0        0.00
3 0  0 0 5        0.00
> orange.rf2.confusion4=table(orange$Country,predict(orange.rf2.bot7,type="respons
e"))
> 100*sum(diag(orange.rf2.confusion4))/sum(orange.rf2.confusion4)
[1] 94.44444
```

This was the last model I made before accuracy began to drop. With the bottom 7 variables (leaving only Calcium and Barium) cut out you still obtain the same accuracy as the full model with a prediction accuracy of 94.44%. This heavily out performs the previously obtained decision tree model that had an accuracy of 63.89%. Of course, that was just used as a baseline model to make sure the random forest was on the right track and not predicting terribly. Given that this model with 2 variables (Calcium and Barium seem to be the only important variables.) is able to predict with a 94.44% accuracy the country of origin an orange comes from; I don't see the need to go back and improve the decision tree as this is a perfectly adequate model.

3. I have placed two datasets in the Canvas web site and in the class directory in SAS. They are **mull2.csv**, and **newvalid.csv**. The last of these files is a validation or test dataset that was collected to evaluate the predictive methodology that we were proposing for invasive plant species in Lava Beds National Monument. The purpose of the analyses is to fit the different classification methods to the "training" data in **mull2.csv** and then evaluate the predictive accuracy of those classifications using 10-fold crossvalidation and by predicting onto the "test" dataset, **newvalid.csv**.

   The dataset **mull2.csv** contains data for 6,048 $30mm \times 30mm$ sites in Lava Beds NM at which common mullein (*Vebascum thapsus*) was detected, and 12,096 of randomly selected "pseudo-absence" sites. That is, randomly selected $30mm \times 30mm$ sites at which mullein is *assumed* to *not* be present. The variable **VETH** identifies these sites as 1 for a presence and 0 for an absence. The dataset **newvalid.csv** contains data from 1,512 randomly selected $30mm \times 30mm$ sites in Lava Beds NM and, again, **VETH** identifies these sites as 1 for a presence of mullein and 0 for an absence. In both these datasets there are topographic variables, *elevation* (in m), *slpd* (slope in degrees), *aspd* (aspect in degrees), *TransAspd* (transformed aspect); three variables giving distances to the nearest roads and trails; and many bioclimatic variables including *relative humidity*, *precipitation*, *temperatures*, and *vapor pressure*. As in the lichen data sets I have carried out a preliminary principal components analysis. For the bioclimatic predictors, a variable name ending in "a" denotes the average of 12 monthly values and the variable name ending in "d" denotes a summer-to-winter contrast.

```
# # # # #
# # # # #
# # 3 # #
# # # # #
# # # # #

mull = read.csv('mull2.csv')
test = read.csv('newvalid.csv')

# discarded LABEGRD_ID, aspd, UTMX, and UTMY within both datasets
# as they are not needed for this analysis.
```

a) Fit a logistic regression model, with and without variable selection, using the dataset **mull2.csv** as the training data and obtain cross-validated accuracies and accuracies for prediction onto the **newvalid.csv** dataset. These are your benchmarks for classification accuracy in this exercise.

Logistic Regression baseline – no variable selection:

```
# logistic regression baseline - no variable selection
mull.lr = glm(VETH~ . ,family=binomial,data=mull)

mull.lr.xval=rep(0,nrow(mull))
xvs=rep(1:10,length=nrow(mull))
xvs=sample(xvs)
for(i in 1:10){
  train=mull[xvs!=i,]
  test=mull[xvs==i,]
  glub=glm(VETH~ . ,family=binomial,data=train)
  mull.lr.xval[xvs==i]=predict(glub,test,type="response")
}
table(mull$VETH,round(mull.lr.xval))
class.sum(mull$VETH,mull.lr.xval)

val.lr = predict(mull.lr,val,type="response")
table(val$VETH,round(val.lr))
class.sum(val$VETH,val.lr)
```

10-fold CV on the training data:

```
> table(mull$VETH,round(mull.lr.xval))

       0     1
  0 10616  1478
  1  2284  3763
> class.sum(mull$VETH,mull.lr.xval)
 [,1]                              [,2]
 "Percent Correctly Classified = " "79.26"
 "Specificity = \n"                "87.78"
 "Sensitivity = "                  "62.23"
 "Kappa ="                         "0.5173"
 "AUC= "                           "0.8457"
```

This model performed decently. It obtained a 79.26% accuracy on the training data. Without anything to compare it to right now, it is a fine baseline.

Validation Set:

```
> val.lr = predict(mull.lr,val,type="response")
> table(val$VETH,round(val.lr))

      0    1
  0 1170  197
  1   25  120
> class.sum(val$VETH,val.lr)
 [,1]                                [,2]
 "Percent Correctly Classified = " "85.32"
 "Specificity = \n"                "85.59"
 "Sensitivity = "                  "82.76"
 "Kappa ="                         "0.4467"
 "AUC= "                           "0.893"
```

It's performed much better than the previous 10-fold. This model achieved an accuracy of 85.32%. Next let's move to variable selection.

Logistic Regression baseline – with variable selection:

Since this is just a baseline, I will just focus on the output of these models rather than the whole process like homework 3.

```
# logistic regression baseline - with variable selection
mull.lr14 = step(mull.lr)

mull.lr14.xval=rep(0,nrow(mull))
xvs=rep(1:10,length=nrow(mull))
xvs=sample(xvs)
for(i in 1:10){
  train=mull[xvs!=i,]
  test=mull[xvs==i,]
  glub=step(glm(VETH~ . ,family=binomial,data=train))
  mull.lr14.xval[xvs==i]=predict(glub,test,type="response")
}
table(mull$VETH,round(mull.lr14.xval))
class.sum(mull$VETH,mull.lr14.xval)

val.lr14 = predict(mull.lr14,val,type="response")
table(val$VETH,round(val.lr14))
class.sum(val$VETH,val.lr14)
```

10-fold CV on the training data:

```
> table(mull$VETH,round(mull.lr14.xval))

       0     1
  0 10604  1490
  1  2293  3754
> class.sum(mull$VETH,mull.lr14.xval)
 [,1]                                [,2]
 "Percent Correctly Classified = " "79.15"
 "Specificity = \n"                "87.68"
 "Sensitivity = "                  "62.08"
 "Kappa ="                         "0.5147"
 "AUC= "                           "0.8452"
```

This model performed on par with the all the previously obtained metrics with no variable selection. It obtained an accuracy of 79.15%, in between the training and validation accuracies. Everything else was pretty similar.

Validation Set:

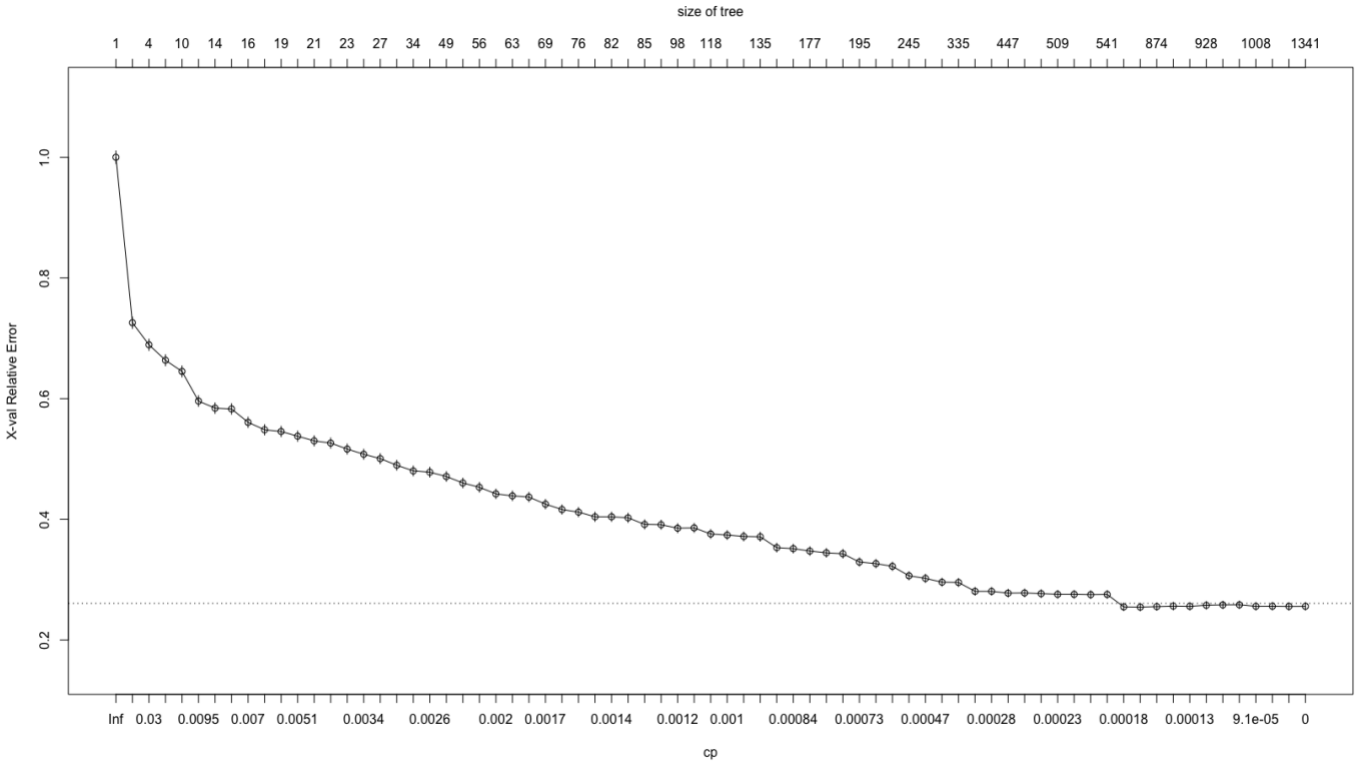
```
> val.lr14 = predict(mull.lr14,val,type="response")
> table(val$VETH,round(val.lr14))

      0    1
  0 1173  194
  1   26  119
> class.sum(val$VETH,val.lr14)
 [,1]                                [,2]
 "Percent Correctly Classified = " "85.45"
 "Specificity = \n"                "85.81"
 "Sensitivity = "                  "82.07"
 "Kappa ="                         "0.4472"
 "AUC= "                           "0.8951"
```

This was the best validation baseline model computed and what I will be using as the baseline for the coming analysis. It obtained an accuracy of 85.45%, higher than the other models, as well as a specificity of 85.81% and sensitivity of 82.07%. If any model is able to obtain a higher accuracy, we will default to that as the new baseline until the best model we can make is obtained.

b) Fit a bunch of different classifiers to the **mull2.csv** data (including classification trees, random forests, adaboost, gradient boosting machines, and support vector machines, the last two with and without tuning) and evaluate their accuracies by cross-validation and on the newvalid.csv data.

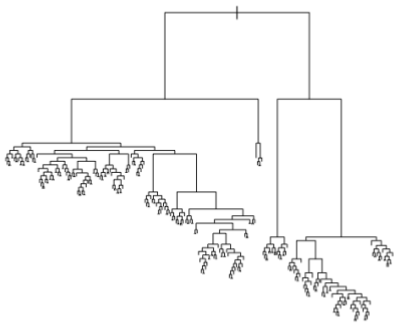
Classification tree:

```
# classification tree
# cp plot
mull.rpartfull=rpart(VETH~ . ,method="class",
                     control=rpart.control(cp=0.0,minsplit=2),data=mull)
plot(mull.rpartfull)
plotcp(mull.rpartfull)
```



I feel like following the 1-SE rule will make the tree outfit and it won't generalize to the validation set. Instead, I will go for a tiny bit above the 1-SE rule, I don't feel like totally ignoring it will be a good idea. The CP value for the classification tree I have determined will be 0.00028. Given that value I will now build the tree.

```
# fitting cp 0.00028
mull.rpartcp028=rpart(VETH~ .
                      ,method="class",data=mull,control=rpart.control(cp=0.00028))
plot(mull.rpartcp028,margin=0.1)

xvs=rep(c(1:10),length=nrow(mull))
xvs=sample(xvs)
mull.rpartcp028.xval=rep(0,length(nrow(mull)))
for(i in 1:10){
  train=mull[xvs!=i,]
  test=mull[xvs==i,]
  rp=rpart(VETH~ .
           ,method="class",data=train,control=rpart.control(cp=0.00028))
  mull.rpartcp028.xval[xvs==i]=predict(rp,test,type="prob")[,2]
}
table(mull$VETH,round(mull.rpartcp028.xval))
class.sum(mull$VETH,mull.rpartcp028.xval)
table(val$VETH,predict(mull.rpartcp028,val,type="class"))
class.sum(val$VETH,predict(mull.rpartcp028,val,type="prob")[,2]
)
```



10-fold CV on training:

```
> table(mull$VETH,round(mull.rpartcp028.xval))

        0     1
  0 11064  1030
  1   910  5137
> class.sum(mull$VETH,mull.rpartcp028.xval)
 [,1]                              [,2]
 "Percent Correctly Classified = " "89.3"
 "Specificity = \n"                "91.45"
 "Sensitivity = "                  "85"
 "Kappa ="                         "0.7605"
 "AUC= "                           "0.9362"
```

The accuracy on this is surprisingly good considering I went down from the 1-SE rule a bit. It obtained an accuracy of 89.3% on the training data, which is higher than the previous logistic regression models. Given that it performed so well on the CV, validation set should be similar.

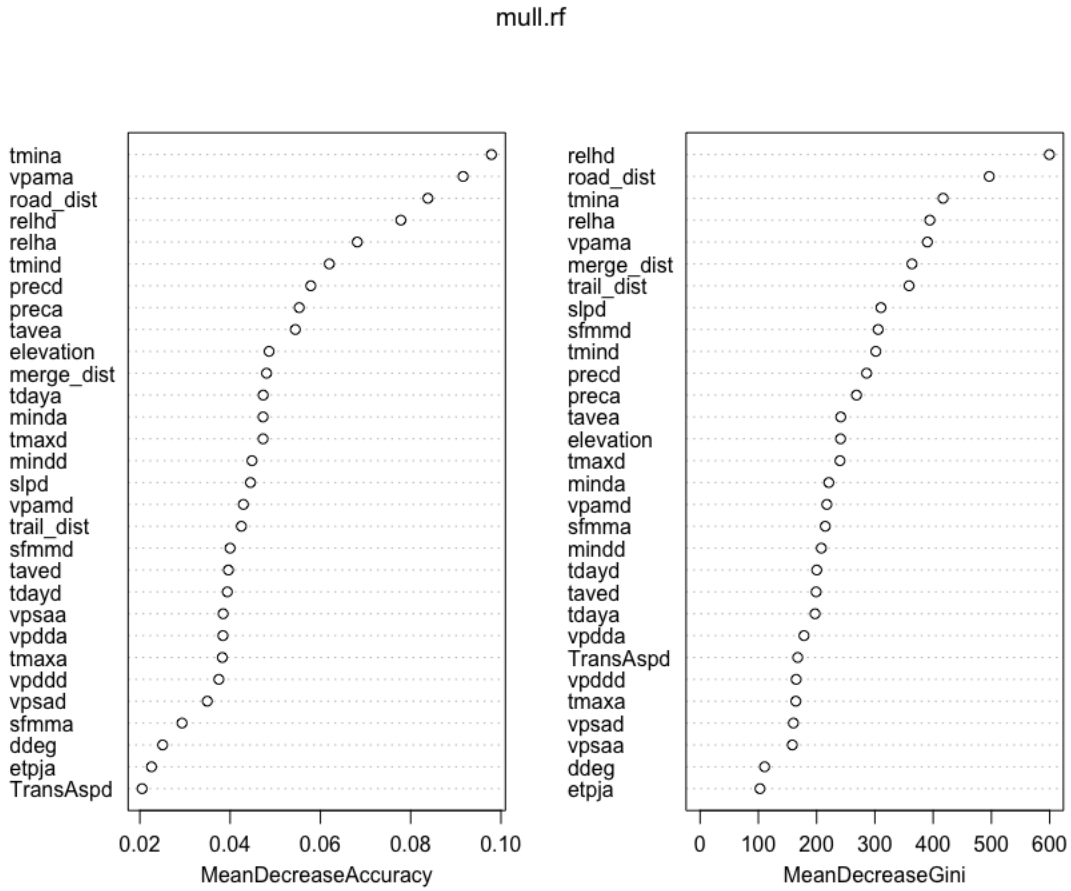
Validation Set:

```
> table(val$VETH,predict(mull.rpartcp028,val,type="class"))

      0    1
  0 1207  160
  1   52   93
> class.sum(val$VETH,predict(mull.rpartcp028,val,type="prob")[,2]
+ )
 [,1]                              [,2]
 "Percent Correctly Classified = " "85.98"
 "Specificity = \n"                "88.3"
 "Sensitivity = "                  "64.14"
 "Kappa ="                         "0.3934"
 "AUC= "                           "0.8385"
```

The validation set actually performed worse than the CV on the training data. This model obtained an accuracy of 85.98% on the validation data. Given that it didn't do so well on the validation data, it seems to be slightly overfit probably due to the huge size. As the decision tree has outperformed the baseline of 85.45%, I will now use this as the new baseline instead of the logistic regression model.

Random Forest Model:

```
# random forest
# varimp plotting
mull.rf=randomForest(as.factor(VETH)~ . ,importance=TRUE,data=mull)
varImpPlot(mull.rf,scale=FALSE)
```



Before I started my random forest analysis I wanted to get a varimp plot and start cutting out variables that aren't important. The reason being that there are a ton of them and there may be a lot of data not needed within the set. From this varimp plot, I want to try taking out vpsaa, vpdda, tmaxa, vpddd, vpsad, sfmma, ddeg, etpja, and TransAspd to start. I will take out more/less depending on how far the accuracy differs from a full random forest and then take the best model and apply it to 10-fold CV and validation data.

Full random forest:

```
# modeling
# full rf
mull.fullrf=randomForest(as.factor(VETH)~ .  ,data=mull)

mull.fullrf$confusion
mull.fullrf.confusion <- table(mull$VETH,predict(mull.fullrf,type="response"))
100*sum(diag(mull.fullrf.confusion))/sum(mull.fullrf.confusion)
```

```
> # modeling
> # full rf
> mull.fullrf=randomForest(as.factor(VETH)~ .  ,data=mull)
> mull.fullrf$confusion
      0    1 class.error
0 11447  647  0.05349760
1   364 5683  0.06019514
> mull.fullrf.confusion <- table(mull$VETH,predict(mull.fullrf,type="response"))
> 100*sum(diag(mull.fullrf.confusion))/sum(mull.fullrf.confusion)
[1] 94.42699
```

The accuracy I'm aiming for with variables cut out is around 94.43%.

Initial variables cut-out model:

```
# first model (bot9)
mull.rfbot9=randomForest(as.factor(VETH)~ .-vpsaa-vpdda-tmaxa-vpddd-vpsad-sfmma-ddeg-etpja-TransAspd,
                         data=mull)

mull.rfbot9$confusion
mull.rfbot9.confusion <- table(mull$VETH,predict(mull.rfbot9,type="response"))
100*sum(diag(mull.rfbot9.confusion))/sum(mull.rfbot9.confusion)
```

```
> # first model (bot9)
> mull.rfbot9=randomForest(as.factor(VETH)~ .-vpsaa-vpdda-tmaxa-vpddd-vpsad-sfmma-dd
eg-etpja-TransAspd,
+                          data=mull)
> mull.rfbot9$confusion
      0    1 class.error
0 11463  631  0.05217463
1   356 5691  0.05887217
> mull.rfbot9.confusion <- table(mull$VETH,predict(mull.rfbot9,type="response"))
> 100*sum(diag(mull.rfbot9.confusion))/sum(mull.rfbot9.confusion)
[1] 94.55929
```

The accuracy on this is actually higher with taking out 9 variables. This model obtained an accuracy of 94.56%. Given that it hasn't dipped and actually gone up, I will take out some more variables in addition to the 9 listed. I will also take out taved and tdayd.

Second variables cut-out model:

```
# second model (bot11)
mull.rfbot11=randomForest(as.factor(VETH)~ .-vpsaa-vpdda-tmaxa-vpddd-vpsad-sfmma-ddeg-etpja-TransAspd
                          -taved-tdayd,
                        data=mull)

mull.rfbot11$confusion
mull.rfbot11.confusion <- table(mull$VETH,predict(mull.rfbot11,type="response"))
100*sum(diag(mull.rfbot11.confusion))/sum(mull.rfbot11.confusion)
```

```
> # second model (bot11)
> mull.rfbot12=randomForest(as.factor(VETH)~ .-vpsaa-vpdda-tmaxa-vpddd-v
psad-sfmma-ddeg-etpja-TransAspd
+                           -taved-tdayd,
+                         data=mull)
> mull.rfbot12$confusion
      0    1 class.error
0 11472  622  0.05143046
1   359 5688  0.05936828
> mull.rfbot12.confusion <- table(mull$VETH,predict(mull.rfbot12,type="r
esponse"))
> 100*sum(diag(mull.rfbot12.confusion))/sum(mull.rfbot12.confusion)
[1] 94.59236
```

It seems like the bottom 11 variables when cut out is the sweet spot, I went above it and accuracy no longer increased and started to drop instead. This model obtained an accuracy of 94.59%. I will now use this model for 10-fold CV and testing on the validation set. Going forward with my analysis I will now exclude the 11 variables selected (taved, tdayd, vpsaa, vpdda, tmaxa, vpddd, vpsad, sfmma, ddeg, etpja, and TransAspd) as they are not important to the analysis (at least I hope I interpreted that correctly and can do that…).

```
# CV and val testing
mull.rfbot11.xval.class=rep(0,length=nrow(mull))
mull.rfbot11.xval.prob=rep(0,length=nrow(mull))
xvs=rep(1:10,length=nrow(mull))
xvs=sample(xvs)
for(i in 1:10){
  train=mull[xvs!=i,]
  test=mull[xvs==i,]
  glub=randomForest(as.factor(VETH)~ .-vpsaa-vpdda-tmaxa-vpddd-vpsad-sfmma
                    -ddeg-etpja-TransAspd-taved-tdayd,
                    data=train)
  mull.rfbot11.xval.class[xvs==i]=predict(glub,test,type="response")
  mull.rfbot11.xval.prob[xvs==i]=predict(glub,test,type="prob")[,2]
}

table(mull$VETH,mull.rfbot11.xval.class)
class.sum(mull$VETH,mull.rfbot11.xval.prob)

table(val$VETH,predict(mull.rfbot11,val,type="response"))
class.sum(val$VETH,predict(mull.rfbot11,val,type="prob")[,2])
```

10-fold CV:

```
> table(mull$VETH,mull.rfbot11.xval.class)
   mull.rfbot11.xval.class
        1     2
  0 11434   660
  1   381  5666
> class.sum(mull$VETH,mull.rfbot11.xval.prob)
 [,1]                                [,2]
 "Percent Correctly Classified = " "94.24"
 "Specificity = \n"                "94.5"
 "Sensitivity = "                  "93.72"
 "Kappa ="                         "0.8719"
 "AUC= "                           "0.9836"
```

I'm fairly optimistic going into the validation since the 10-fold CV accuracy was 94.24%. It seems like this model should perform much better than the previous decision tree model.

Validation data:

```
> table(val$VETH,predict(mull.rfbot11,val,type="response"))

      0    1
  0 1265  102
  1   46   99
> class.sum(val$VETH,predict(mull.rfbot11,val,type="prob")[,2])
 [,1]                                [,2]
 "Percent Correctly Classified = " "90.15"
 "Specificity = \n"                "92.47"
 "Sensitivity = "                  "68.28"
 "Kappa ="                         "0.5166"
 "AUC= "                           "0.8661"
```

On the validation data it was roughly 4% down from the 10-fold CV on training data. I thought it would've been a bit closer, but this is still a very good model. With the decision tree on the validation data, I was only able to get 85.98% accuracy. With this random forest model, I'm able to get an accuracy of 90.15%. Since it heavily outperformed the decision tree, I will now use this as the baseline accuracy needed going forwards. Next is trying this data on adaboost.

Adaboosting:

```
# adaboost
library(ada)

mull.ada=ada(as.factor(VETH)~ .-vpsaa-vpdda-tmaxa-vpddd-vpsad-sfmma
             -ddeg-etpja-TransAspd-taved-tdayd
             ,loss="exponential",data=mull)

mull.ada.xvalpr=rep(0,nrow(mull))
xvs=rep(1:10,length=nrow(mull))
xvs=sample(xvs)
for(i in 1:10){
  train=mull[xvs!=i,]
  test=mull[xvs==i,]
  glub=ada(as.factor(VETH)~ .-vpsaa-vpdda-tmaxa-vpddd-vpsad-sfmma
           -ddeg-etpja-TransAspd-taved-tdayd
           ,loss="exponential",data=train)
  mull.ada.xvalpr[xvs==i]=predict(glub,newdata=test,type="prob")[,2]
}

table(mull$VETH,round(mull.ada.xvalpr))
class.sum(mull$VETH,mull.ada.xvalpr)

table(val$VETH,round(predict(mull.ada,newdata=val,type="prob")[,2]))
class.sum(val$VETH,predict(mull.ada,newdata=val,type="prob")[,2])
```

For the adaboost model I have decided to cut out the same 11 variables from the random forest model.

10-fold CV:

```
> table(mull$VETH,round(mull.ada.xvalpr))

       0     1
  0 10932  1162
  1  1825  4222
> class.sum(mull$VETH,mull.ada.xvalpr)
 [,1]                              [,2]
 "Percent Correctly Classified = " "83.53"
 "Specificity = \n"                "90.39"
 "Sensitivity = "                  "69.82"
 "Kappa ="                         "0.6191"
 "AUC= "                           "0.908"
```

I have a feeling that this model will not outperform the random forest just computed. For the adaboost within the 10-fold CV it obtained an accuracy of 83.53%. Unless a miracle happens, this will not outperform either the decision tree or random forest on the validation set.

Validation set:

```
> table(val$VETH,round(predict(mull.ada,newdata=val,t
ype="prob")[,2]))

       0    1
  0 1180  187
  1   28  117
> class.sum(val$VETH,predict(mull.ada,newdata=val,typ
e="prob")[,2])
 [,1]                                [,2]
 "Percent Correctly Classified = " "85.78"
 "Specificity = \n"                "86.32"
 "Sensitivity = "                  "80.69"
 "Kappa ="                         "0.4497"
 "AUC= "                           "0.9075"
```

It did perform a bit better than the 10-fold CV. But with an accuracy of 85.78% it is nowhere near performing like the random forest did on this data. Since it didn't outperform the current baseline of 90.15% on the validation data, I will reject this model and keep the current baseline in the random forest model.

Gradient boosting:

```
# gradient boosting machine
library(caret)

fitControl = trainControl(method = "cv", number = 10)

# tuning
gbmGrid = expand.grid(interaction.depth = c(12, 14, 16, 18, 20, 22, 24),
                      n.trees = c(25,50,75,100,125,150), shrinkage = c(0.01, 0.05, 0.1, 0.2, 0.25, 0.3),
                      n.minobsinnode=10)
gbmFit = train(as.factor(VETH)~ .-vpsaa-vpdda-tmaxa-vpddd-vpsad-sfmma
               -ddeg-etpja-TransAspd-taved-tdayd,
               method="gbm", tuneGrid = gbmGrid, trControl = fitControl, data=mull)
gbmFit
```

```
Tuning parameter 'n.minobsinnode' was held constant at
 a value of 10
Accuracy was used to select the optimal model using
 the largest value.
The final values used for the model were n.trees =
 150, interaction.depth = 24, shrinkage = 0.3
 and n.minobsinnode = 10.
```

After tuning, it was determined that the optimal model from the values passed through is n. trees = 150, interaction depth = 24, shrinkage = 0.3, and n.minobsinnode = 10. Now that those values have been obtained, let's apply them to 10-fold CV and the validation data.

```
# modeling
mull.gbm2=gbm(VETH ~ .-vpsaa-vpdda-tmaxa-vpddd-vpsad-sfmma
              -ddeg-etpja-TransAspd-taved-tdayd,
              distribution="bernoulli",interaction.depth=24,
              n.trees=150, shrinkage=0.3,n.minobsinnode=10,data=mull)

mull.gbmopt.xvalpr=rep(0,nrow(mull))
xvs=rep(1:10,length=nrow(mull))
xvs=sample(xvs)
for(i in 1:10){
  train=mull[xvs!=i,]
  test=mull[xvs==i,]
  glub=gbm(VETH~ . -vpsaa-vpdda-tmaxa-vpddd-vpsad-sfmma
           -ddeg-etpja-TransAspd-taved-tdayd,
           distribution="bernoulli",interaction.depth=24,n.trees=150,
           shrinkage=0.3,n.minobsinnode=10,data=train)
  mull.gbmopt.xvalpr[xvs==i]=predict(glub,newdata=test,type="response",n.trees=50)
}

table(mull$VETH,round(mull.gbmopt.xvalpr))
class.sum(mull$VETH,mull.gbmopt.xvalpr)

table(val$VETH,round(predict(mull.gbm2,newdata=val,type="response",n.trees=150)))
class.sum(val$VETH,predict(mull.gbm2,newdata=val,type="response",n.trees=150))
```

10-fold CV:

```
> table(mull$VETH,round(mull.gbmopt.xvalpr))

        0     1
  0 11229   865
  1   852  5195
> class.sum(mull$VETH,mull.gbmopt.xvalpr)
 [,1]                                [,2]
 "Percent Correctly Classified = " "90.54"
 "Specificity = \n"                "92.85"
 "Sensitivity = "                  "85.91"
 "Kappa ="                         "0.7872"
 "AUC= "                           "0.9608"
```

On the outset, gradient boosting is looking good for how it will perform on the validation data. This is a great accuracy for the 10-fold CV. 90.54% is the accuracy obtained by this model. This model may be the new baseline if the validation accuracy is able to follow suit.

Validation set:

```
> table(val$VETH,round(predict(mull.gbm2,newdata=val,type="response",n.trees=150)))

      0    1
  0 1248  119
  1   55   90
> class.sum(val$VETH,predict(mull.gbm2,newdata=val,type="response",n.trees=150))
 [,1]                                [,2]
 "Percent Correctly Classified = " "88.49"
 "Specificity = \n"                "91.29"
 "Sensitivity = "                  "62.07"
 "Kappa ="                         "0.4457"
 "AUC= "                           "0.8369"
```

This model didn't live up to the 10-fold CV estimation. It obtained an accuracy of 88.49%. The baseline of 90.15% on the validation set made using the random forest is still holding up. Unless the SVM is able to perform better it seems like the random forest will be the best model for this data.

SVM:

```
# svm
library(e1071)
library(EZtune)

mullcut = read.csv('mull2cut.csv')
valcut = read.csv('newvalidcut.csv')

xmull = as.matrix(mullcut[,1:20])
ymull = as.vector(mullcut[,21])

mull.svm.tune <- eztune(xmull, ymull,method="svm", fast=FALSE, cross=10)
mull.svm.tune
```

```
> mull.svm.tune
$nfold
[1] 10

$cost
[1] 47.001

$gamma
[1] 31.97783

$loss
[1] 0.949231

$model

Call:
svm(formula = as.factor(y) ~ ., data = dat, cost = results$cost,
    gamma = results$gamma, probability = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  47.001

Number of Support Vectors:  14558


$variables
 [1] "elevation"  "slpd"       "etpjd"      "minda"
 [5] "mindd"      "preca"      "precd"      "relha"
 [9] "relhd"      "sfmmd"      "tavea"      "tdaya"
[13] "tmaxd"      "tmina"      "tmind"      "vpama"
[17] "vpamd"      "road_dist"  "trail_dist" "merge_dist"

$levels
[1] "0" "1"

attr(,"class")
[1] "eztune"
```

The training took quite a bit for the SVM. I ran it overnight so I don't have exactly how long it took, but it was around ~9 hours, started around 9 pm, finished around 4:30 am. I probably should have randomly sampled 50% of the data for use in the tuning process like the EZtune documentation says to do on the website. From this tuning I will be using a cost of 47.001 and a gamma of 31.97783. (I'll be slightly annoyed if after all that training this model performs worse than the random forest, but we'll see.)

```
mull.tunedsvm=svm(as.factor(VETH)~ .,
                  probability=TRUE, cost=47.001, gamma=31.97783, data=mullcut)

mull.tunedsvm.xvalpred=rep(0,nrow(mullcut))
xvs=rep(1:10,length=nrow(mullcut))
xvs=sample(xvs)
for(i in 1:10){
  train=mullcut[xvs!=i,]
  test=mullcut[xvs==i,]
  glub=svm(as.factor(VETH)~ .,
           probability=TRUE, cost = 47.001, gamma = 31.97783, data=train)
  mull.tunedsvm.xvalpred[xvs==i]=attr(predict(glub,test,probability=TRUE),"probabilities")[,1]
}

table(mullcut$VETH,round(mull.tunedsvm.xvalpred))
class.sum(mullcut$VETH,mull.tunedsvm.xvalpred)

mull.tunedsvm.valpred=predict(mull.tunedsvm,valcut,probability=TRUE)
table(valcut$VETH,round(attr(mull.tunedsvm.valpred,"probabilities")[,1]))
class.sum(valcut$VETH,attr(mull.tunedsvm.valpred,"probabilities")[,1])
```

10-Fold CV:

```
> table(mullcut$VETH,round(mull.tunedsvm.xvalpred))

      0    1
  0 11711  383
  1   583 5464
> class.sum(mullcut$VETH,mull.tunedsvm.xvalpred)
 [,1]                                [,2]
 "Percent Correctly Classified = " "94.68"
 "Specificity = \n"                "96.83"
 "Sensitivity = "                  "90.36"
 "Kappa ="                         "0.8792"
 "AUC= "                           "0.9582"
```

This is a fairly good model, achieving an accuracy of 94.68%. It's looking good for a contender to take down the previous random forest model that I currently the highest performing model at 90.15% on the validation set. Both specificity and sensitivity are also tight within this model, there isn't a ton of spread between the two.

Validation set:

```
> table(valcut$VETH,round(attr(mull.tunedsvm.valpred,"probabilities")[,1]))

      0    1
  0 1274   93
  1   59   86
> class.sum(valcut$VETH,attr(mull.tunedsvm.valpred,"probabilities")[,1])
 [,1]                              [,2]
 "Percent Correctly Classified = " "89.95"
 "Specificity = \n"                "93.2"
 "Sensitivity = "                  "59.31"
 "Kappa ="                         "0.4753"
 "AUC= "                           "0.7652"
```

The SVM performed pretty well on the validation set. Sadly, it didn't perform nearly as well as the random forest model. This model obtained an accuracy of 89.95%, which is the second best performing model thus far, just shy of the random forest model (90.15%). The major problem with this model is that sensitivity is performing poorly overall. While the CV had the difference between sensitivity and specificity really close, with this, sensitivity is extremely low.

c) Summarize (e.g., a table or two) and discuss your analyses. You should include some discussion of which variables are important to the classification and what the nature of their relationships are to the probability of finding mullein.

Model on 10-fold CV ->	Logreg – No Variable Selection	Logreg – Variable Selection	Decision Tree	Random Forest	Adaboost	Gradient Boosting	SVM
Accuracy	79.26%	79.15%	89.30%	94.24%	83.53%	90.54%	94.68%
Specificity	87.78%	87.68%	91.45%	94.50%	90.39%	92.85%	96.83%
Sensitivity	62.23%	62.08%	85.00%	93.72%	69.82%	85.91%	90.36%
Model on validation set ->	Logreg – No Variable Selection	Logreg – Variable Selection	Decision Tree	Random Forest	Adaboost	Gradient Boosting	SVM
Accuracy	85.32%	85.45%	85.98%	90.15%	85.78%	88.49%	89.95%
Specificity	85.59%	85.81%	88.30%	92.47%	86.32%	91.29%	93.20%
Sensitivity	82.76%	82.07%	64.14%	68.28%	80.69%	62.07%	59.31%

It seems that the random forest model performed the best on this data. Using the random forest model, it obtained the highest accuracy with a 90.15% on the validation set. If you wanted the specificity and sensitivity to be much closer though I would go with the Adaboost model. While it only has an accuracy of 85.78%, the spread between specificity and sensitivity is much less (Adaboost: 86.32% - specificity, 80.69% - sensitivity; Random Forest: 92.47% - specificity, 64.14% - sensitivity).

As for which variables seemed to be the most important for this analysis on the probability of finding mullein (as determined by the VarImp plot). It seemed that the four most important variables were tmina, vpama, road_dist, and relhd. The unneeded variables for analysis were found to be taved, tdayd, vpsaa, vpdda, tmaxa, vpddd, vpsad, sfmma, ddeg, etpja, and TransAspd. By excluding them there was no negative effect on the analysis. In total there were 20 variables left in for modeling, they were elevation, slpd, etpjd, minda, mindd, preca, relha, relhd, sfmmd, tavea, tdaya, tmaxd, timna, tmind, vpama, vpamd, road_dist, trail_dist, and merge_dist.

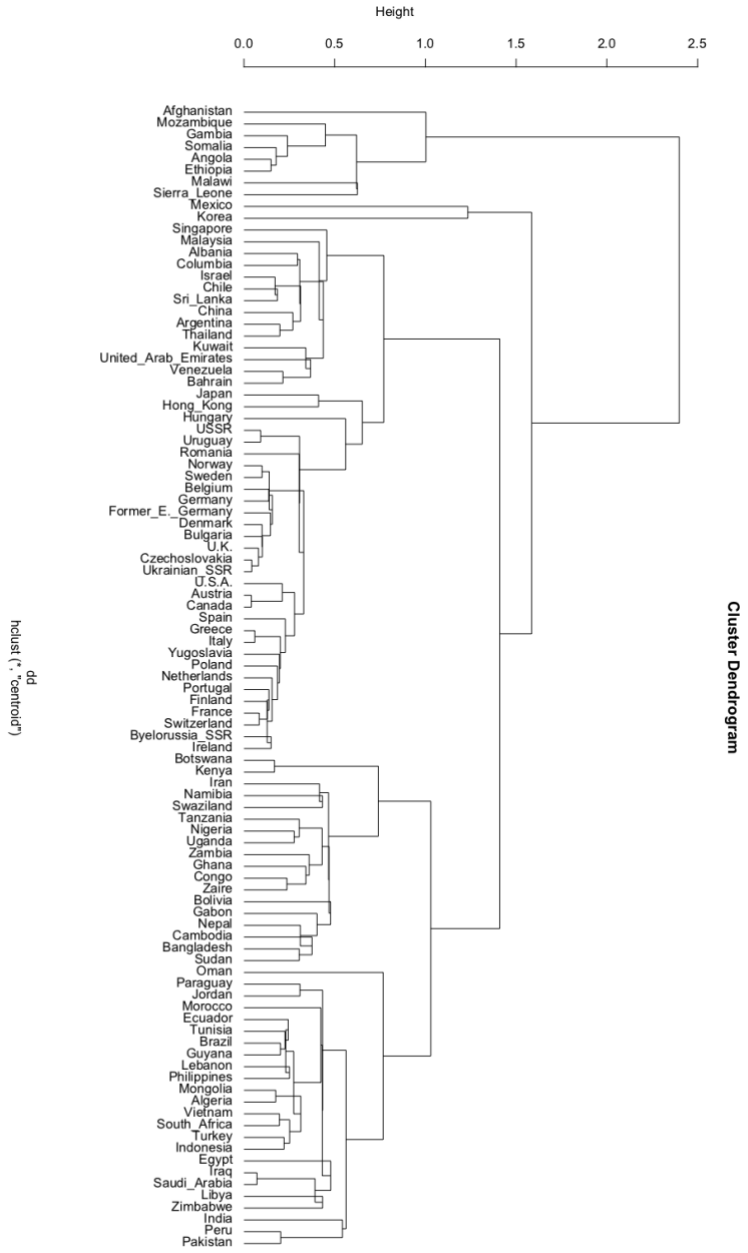
To the nature of the relationship for the probability of finding mullein it is as follows by looking at the Mean Decrease Accuracy in order of greatest to least: tmina – 0.10, vpama – 0.09, road_dist – 0.085, relhd – 0.08, relha – 0.072, tmind – 0.066, precd – 0.06, preca – 0.059, tavea – 0.058, elevation – 0.05, merge_dist – 0.05, tdaya – 0.049, minda – 0.049, tmaxd – 0.049, mindd – 0.047, slpd – 0.046, vpamd – 0.44, trail_dist – 0.043, sfmmd – 0.041. We can see the high Mean Decrease Accuracy drops off fairly quick after the top 4 meaning that in terms of the probability of finding mullein tmina, vpama, road_dist, and relhd are the most important to look at.

4. For this question I have uploaded into Canvas and SAS the dataset **Poverty.cvs** (and **Poverty.xlsx**). The data concerns births, deaths, and infant deaths for many countries. Apply cluster analysis to these data and report on your results. Are there any groupings that surprise you?

```
# # # # #
# # # # #
# # 4 # #
# # # # #
# # # # #

library(cluster)
?hclust

pov = read.csv("Poverty.csv")

# cleaning up dataframe for clustering
rownames(pov) = pov$Country
pov = pov[,-4]

dd <- dist(scale(pov), method = "euclidean")
hc.ward <- hclust(dd, method = "ward.D2")
hc.cen <- hclust(dd, method = "cen")
plot(hc.ward, hang = -1)
plot(hc.cen, hang = -1)
```

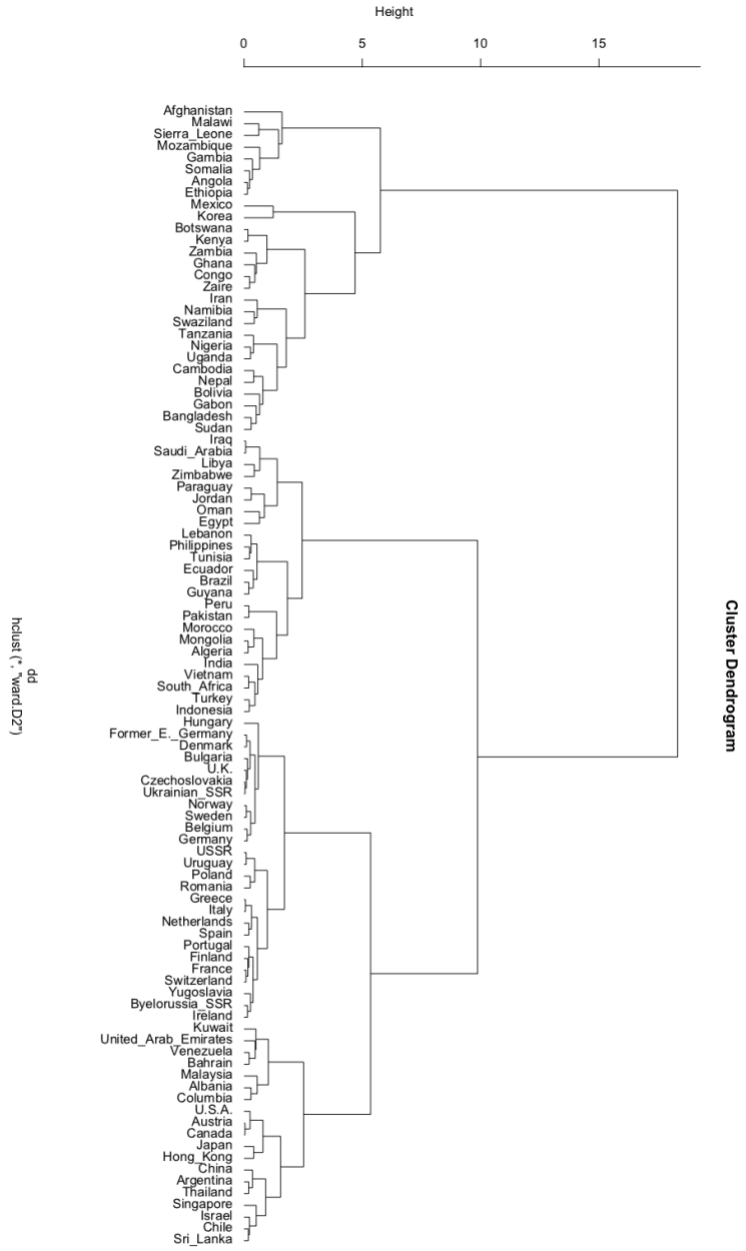
I decided to run through 2 hierarchical approaches, centroid and ward. I'm not going to do k-NN right now due to the fact that the correct number of clusters is unknown.

Centroid:



There are a couple groupings that surprise me within the centroid method. First is USSR and Uruguay. They are so distant from one another that I didn't think they would be grouped right off the bat. Another is Mexico and Korea, somewhat similar reasoning as to why it surprises me, but also, they are two countries are very different culturally. Those are the two surprising groupings to me within the centroid clustering method.

Ward:



Looking at the ward method, there are also a few surprising groupings. UK, Czechoslovakia, and Ukrainian_SSR are all grouped together. The UK is a stand out from the other two, I figured a country with such a high GDP wouldn't be grouped with those two. Canada and Austria are grouped together, again, while both have pretty good winters, they are so different culturally and geographically I would've thought that the grouping would be different. Paraguay and Jordan is another grouping that caught my eye. This is simply because most African countries seemed to be grouped together except Jordan.

5. The data **chemo2.csv** contains data from a large chemical engineering experiment. The first 22 variables (all beginning with "p") are predictor variables. They are temperatures at various places in the reactor. The last 6 variables (all of which begin with an "l") are response variables, and are measures of the production of the chemical reactor.

```
# # # # #
# # # # #
# # 5 # #
# # # # #
# # # # #

chemo = read.csv("chemo2.csv")
chemolr1 = chemo[,-24:-28]
chemo = chemo[,-23]
chemolr2 = chemo[,-24:-27]
chemo = chemo[,-23]
chemolr3 = chemo[,-24:-26]
chemo = chemo[,-23]
chemolr4 = chemo[,-24:-25]
chemo = chemo[,-23]
chemolr5 = chemo[,-24]
chemolr6 = chemo[,-23]
```

Breaking out was just in case it was needed.

Setting up the data

a) Carry out principal components analysis on the predictor variables.

```
# over x
chemoX.eig = eigen(cov(chemoX))
100*chemoX.eig$values/sum(chemoX.eig$values)
```

Over all predictor variables:

```
> # over x
> chemoX.eig = eigen(cov(chemoX))
> 100*chemoX.eig$values/sum(chemoX.eig$values)
 [1] 9.966811e+01 1.674823e-01 7.116533e-02 4.914846e-02 1.642223e-02
 [6] 1.113515e-02 5.450799e-03 3.188185e-03 2.221492e-03 1.885092e-03
[11] 1.383464e-03 9.110732e-04 7.813608e-04 3.707245e-04 2.059197e-04
[16] 8.581954e-05 2.481366e-05 1.249836e-05 4.464815e-06 2.961883e-06
[21] 1.941873e-06 1.206774e-06
```

When looking at all the data and predictors it seems like this is one-dimensional data with 99.67% of the data explained by one dimension.

b) Carry out canonical correlation analysis on the data.

```
# # #
# b #
# # #

chemo = read.csv("chemo2.csv")
chemoX = chemo[,-23:-28]
chemoY = chemo[,-1:-22]

# canonical correlation
library(CCP)
library(CCA)

cc1 <- cc(chemoY, chemoX)
rho <- cc1$cor

n <- dim(chemoY)[1]
p <- length(chemoY)
q <- length(chemoX)

p.asym(rho, n, p, q, tstat = "Wilks")
```

```
> cc1 <- cc(chemoY, chemoX)
> rho <- cc1$cor
> n <- dim(chemoY)[1]
> p <- length(chemoY)
> q <- length(chemoX)
> p.asym(rho, n, p, q, tstat = "Wilks")
Wilks' Lambda, using F-approximation (Rao's F):
                stat    approx df1       df2      p.value
1 to 6:  3.610637e-06 9.8520071 132 170.54586 0.000000e+00
2 to 6:  5.871129e-04 4.9957561 105 146.52274 0.000000e+00
3 to 6:  2.391726e-02 2.3794783  80 120.76746 7.973021e-06
4 to 6:  2.018449e-01 1.1622023  57  93.25801 2.569594e-01
5 to 6:  3.740702e-01 1.1289271  36  64.00000 3.302371e-01
6 to 6:  6.752930e-01 0.9333927  17  33.00000 5.460340e-01
```

For the tstat on this canonical correlation I used Wilks.

It looks like from the canonical correlation you should use 3 canonical correlations as after that the p-values become insignificant (using significance level of 0.05). This is interesting as the PCA seemed to indicate that the data was one dimensional.

c) Summarize and discuss your findings.

Looking at both the PCA and CCA, While I'm not the greatest at explaining CCA (it's still new to me), I would say it is probably better to go with CCA in this case. Using the first three dimensions of canonical correlation would probably be better in explaining the data (as the p-values are lower than the level of significance) than just using the first two dimensions within the PCA. The reason I believe that CCA is the way to go is that is because I prefer using linear combinations within the sets of variables rather than just going for dimensionality reduction. If we already have the data and aren't looking to cut down on collection costs, I believe CCA is going to be greater here. But it depends on the goal you are looking for as they are two different things.