Lab Assignment 5: Quant III

Kyle Davis
In collaberation with Andrew Goodhart.

Question 1:
Creating three DGPs:

```
library(caret)
library(MASS)
library(Rlab)
library(boot)
library(ggplot2)
library(ggridges)
library(dplyr)

set.seed(12345)

N <- 2000
P <- 50

## Ballanced
mu    <- runif(P, -1,1)
Sigma <- rWishart(n=1, df=P, Sigma=diag(P))[,,1]
Sigma <- ifelse(row(Sigma) != col(Sigma), 0, Sigma)
X     <- mvrnorm(N, mu=mu, Sigma = Sigma)
p     <- rbern(P, 1)
beta  <- p*rnorm(P,1,0.9) + (1-p)*rnorm(P,0,0.3)
eta   <- X%*%beta
pi    <- inv.logit(eta)
Y     <- rbern(N, pi)
## sum(Y)/length(Y)
Y     <- as.factor(Y)
data.1.Bal <- data.frame(X, Y)


## High >> .5
mu2    <- runif(P, 0.5, 1.5)
Sigma2 <- rWishart(n=1, df=P, Sigma=diag(P))[,,1]
Sigma2 <- ifelse(row(Sigma2) != col(Sigma2), 0, Sigma2)
X2     <- mvrnorm(N, mu=mu2, Sigma = Sigma2)
p2     <- rbern(P, 1)
beta2  <- p2*rnorm(P,1,0.1) + (1-p2)*rnorm(P,0,1)
eta2   <- X2%*%beta2
pi2    <- inv.logit(eta2)
Y2     <- rbern(N, pi2)
```

```
## sum(Y2)/length(Y2)
Y2     <- as.factor(Y2)
data.2.High <- data.frame(X2, Y2)


## Low << .5
mu3    <- runif(P, -2, 0)
Sigma3 <- rWishart(n=1, df=P, Sigma=diag(P))[,,1]
Sigma3 <- ifelse(row(Sigma3) != col(Sigma3), 0, Sigma3)
X3     <- mvrnorm(N, mu=mu3, Sigma = Sigma3)
p3     <- rbern(P, 1)
beta3  <- p3*rnorm(P,1,1.7) + (1-p3)*rnorm(P,0,0.01)
eta3   <- X3%*%beta3
pi3    <- inv.logit(eta3)
Y3     <- rbern(N, pi3)
## sum(Y3)/length(Y3)
Y3     <- as.factor(Y3)
data.3.Low <- data.frame(X3, Y3)
```

The data generated follow a Bernoulli distribution and are roughly ballanced (data set 1), have a high probability of success (data set 2), and a low probability of success (data set 3).

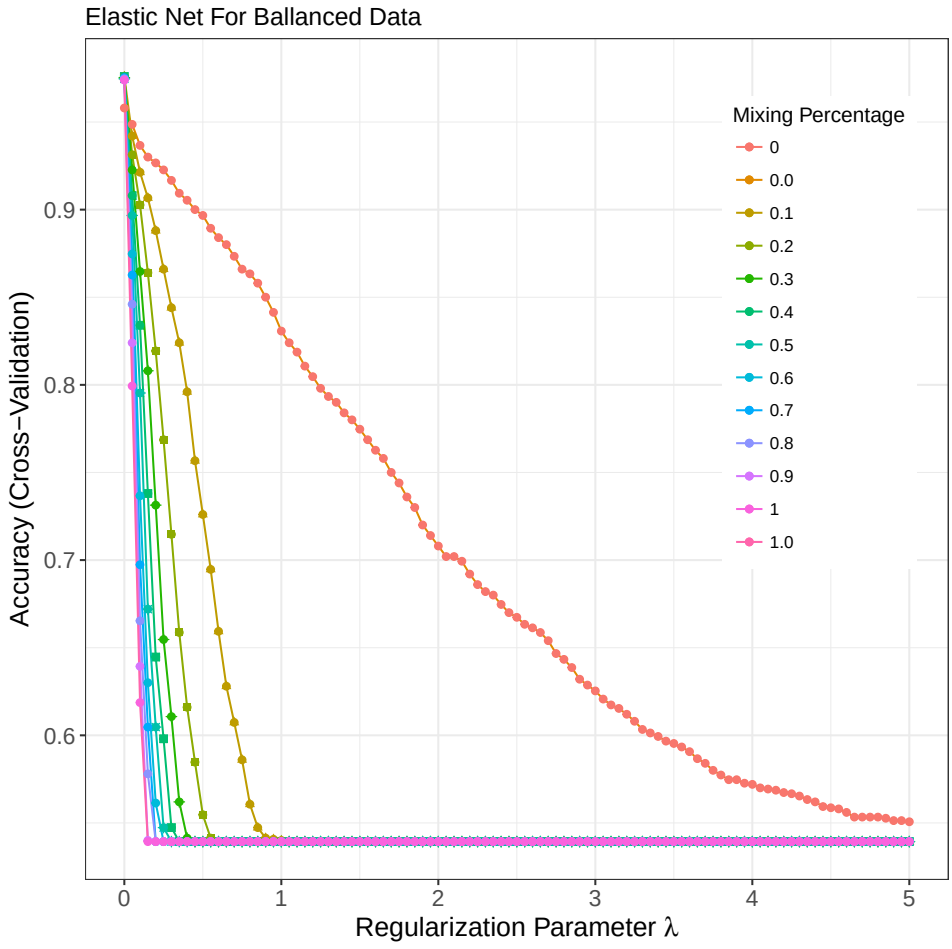
Question 2:
Dividing each data set into test and training sets:

```
## Test/Training Sets for Even Data
test.data.1.Bal   <- data.1.Bal[1:500,]
train.data.1.Bal  <- data.1.Bal[501:2000,]

## Test/Training Sets for High Success Data
test.data.2.High  <- data.2.High[1:500,]
train.data.2.High <- data.2.High[501:2000,]

## Test/Training Sets for Low Success Data
test.data.3.Low   <- data.3.Low[1:500,]
train.data.3.Low  <- data.3.Low[501:2000,]
```

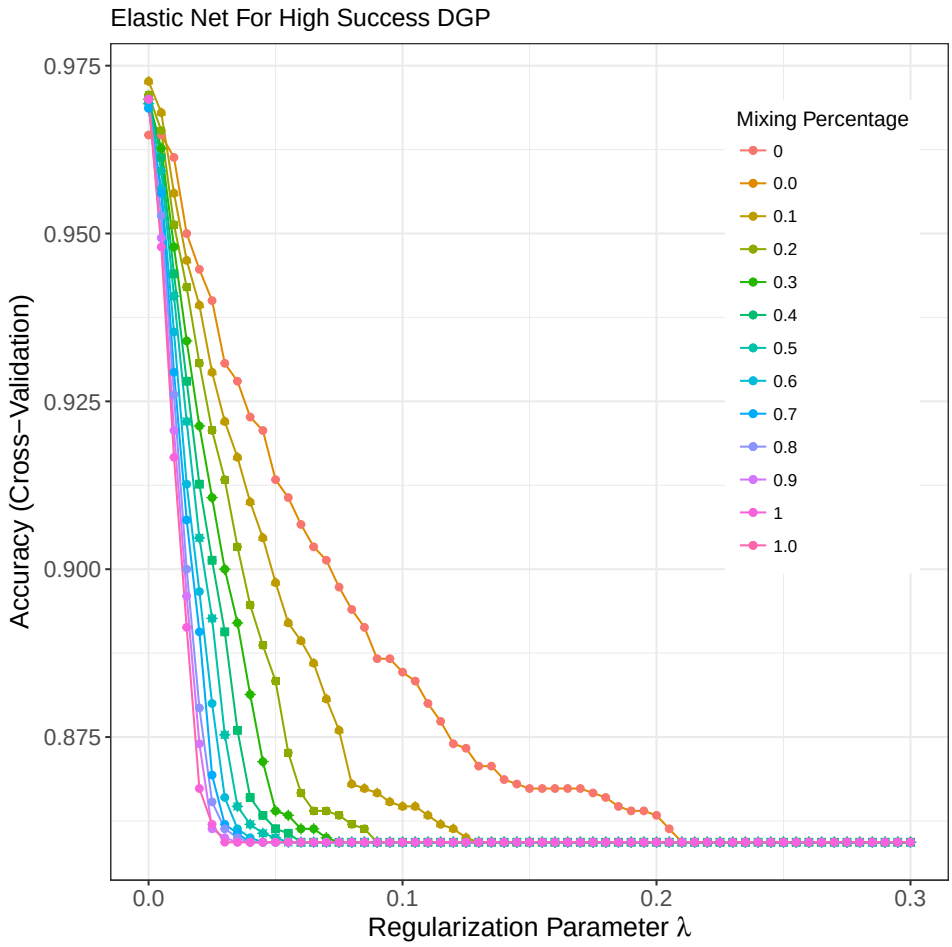
Question 3:

```
### All of these elastic net lambda's for the assignment have been narrowed from 0-200 a

## Elastic Net:
enet.Bal <- train(Y~., method="glmnet",
```

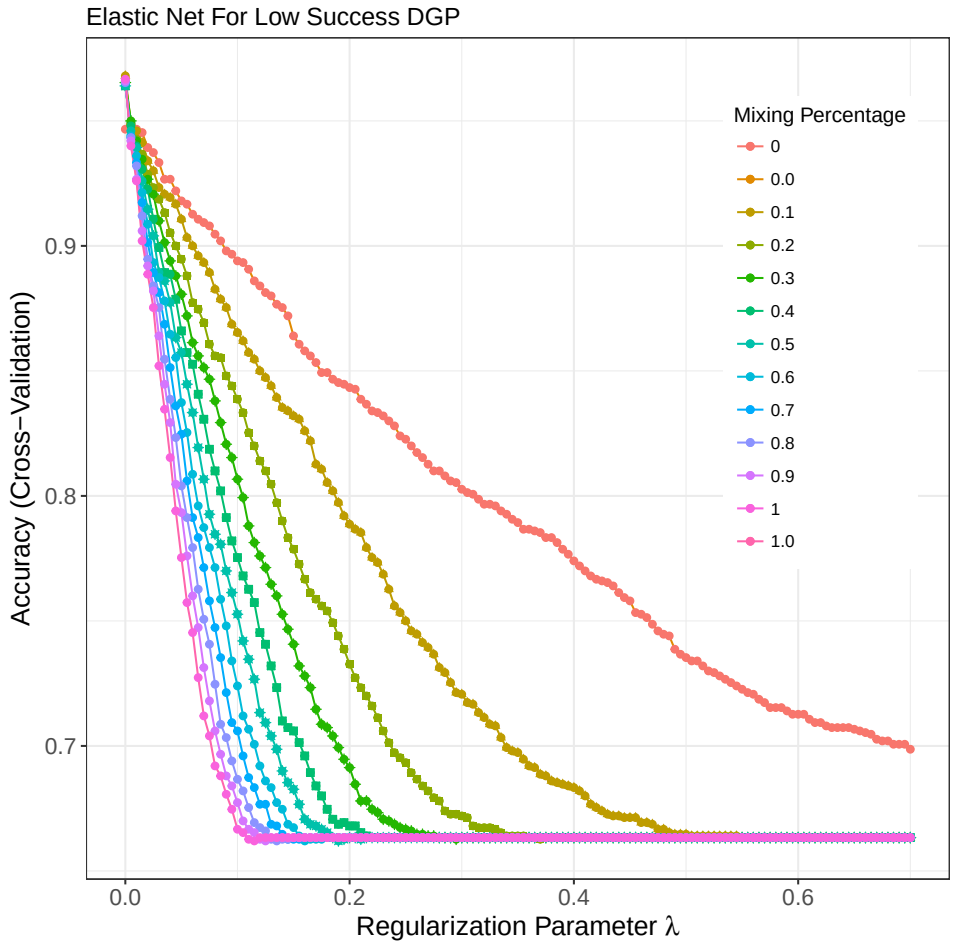
```
                      tuneGrid=expand.grid(alpha=seq(0, 1, .1),
                                           lambda=seq(0, 5,.05)),
                      data=train.data.1.Bal,
                      preProcess=c("center"),
                      trControl=trainControl(method="cv",number=2, search="grid"))

ggplot(enet.Bal, aes(x=enet.Bal$results$lambda,
                      y=enet.Bal$results$Accuracy))+
     geom_point( aes(colour = factor(enet.Bal$results$alpha)) )+
     guides(shape = "none")+
     ggtitle("Elastic Net For Ballanced Data")+
     xlab( expression(paste("Regularization Parameter ", lambda)))+
     theme_bw()+
     theme(axis.text=element_text(size=12),
           axis.title=element_text(size=14))+
     theme(legend.position = c(.85, .65))
```



```r
## High
enet.High <- train(Y2~., method="glmnet",
                   tuneGrid=expand.grid(alpha=seq(0, 1, .1),
                                        lambda=seq(0,.3,.005)),
                   data=train.data.2.High,
                   preProcess=c("center"),
                   trControl=trainControl(method="cv",number=2, search="grid"))

ggplot(enet.High, aes(x=enet.High$results$lambda,
                      y=enet.High$results$Accuracy))+
     geom_point( aes(colour = factor(enet.High$results$alpha)) )+
     guides(shape = "none")+
     ggtitle("Elastic Net For High Success DGP")+
     xlab( expression(paste("Regularization Parameter ", lambda)))+
     theme_bw()+
     theme(axis.text=element_text(size=12),
           axis.title=element_text(size=14))+
     theme(legend.position = c(.85, .65))
```



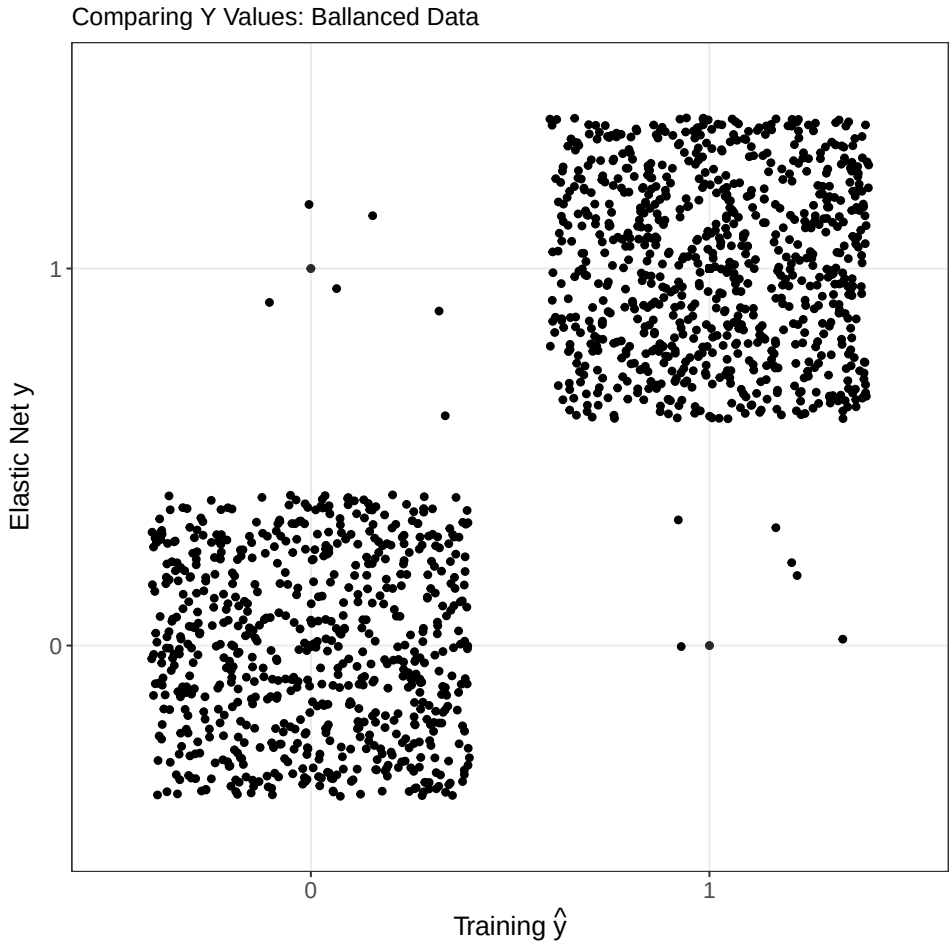
```r
## Low
enet.Low <- train(Y3~., method="glmnet",
                  tuneGrid=expand.grid(alpha=seq(0,  1, .1),
                                       lambda=seq(0, .7, .005)),
                  data=train.data.3.Low,
                  preProcess=c("center"),
                  trControl=trainControl(method="cv",number=2, search="grid"))

ggplot(enet.Low, aes(x=enet.Low$results$lambda,
                     y=enet.Low$results$Accuracy))+
     geom_point( aes(colour = factor(enet.Low$results$alpha)) )+
     guides(shape = "none")+
     ggtitle("Elastic Net For Low Success DGP")+
     xlab( expression(paste("Regularization Parameter ", lambda)))+
     theme_bw()+
     theme(axis.text=element_text(size=12),
           axis.title=element_text(size=14))+
     theme(legend.position = c(.85, .65))
```



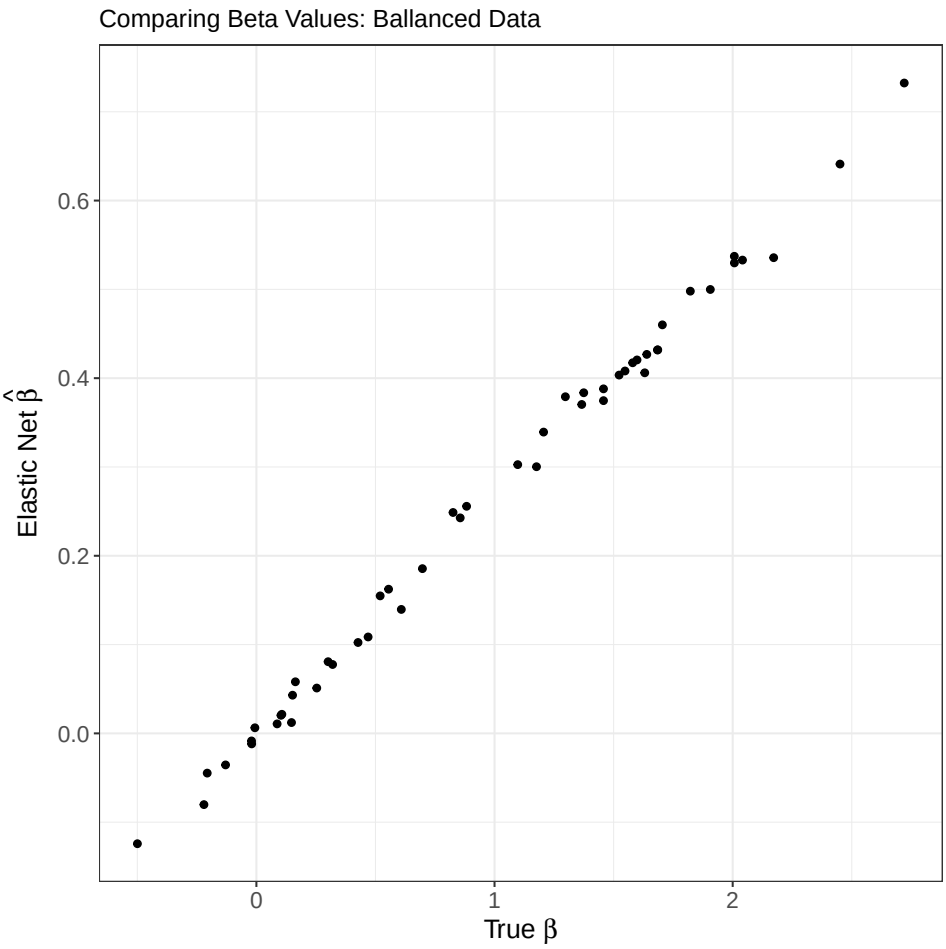
Each of these fabulous graphs show our accuracy when using each alpha parameter and our

resulting best-fit lambda. For each data set our best lambda is zero and for each dataset our best alpha parameter is near zero (the LASSO) our highly predictive data has a best alpha set at .2 whereas our low data has the best alpha at .1; our ballanced data has the best alpha at zero. These Results are useful for noting how our model performs and under what conditions it performs best, but let's see how it performs to the actual DGP that created it:

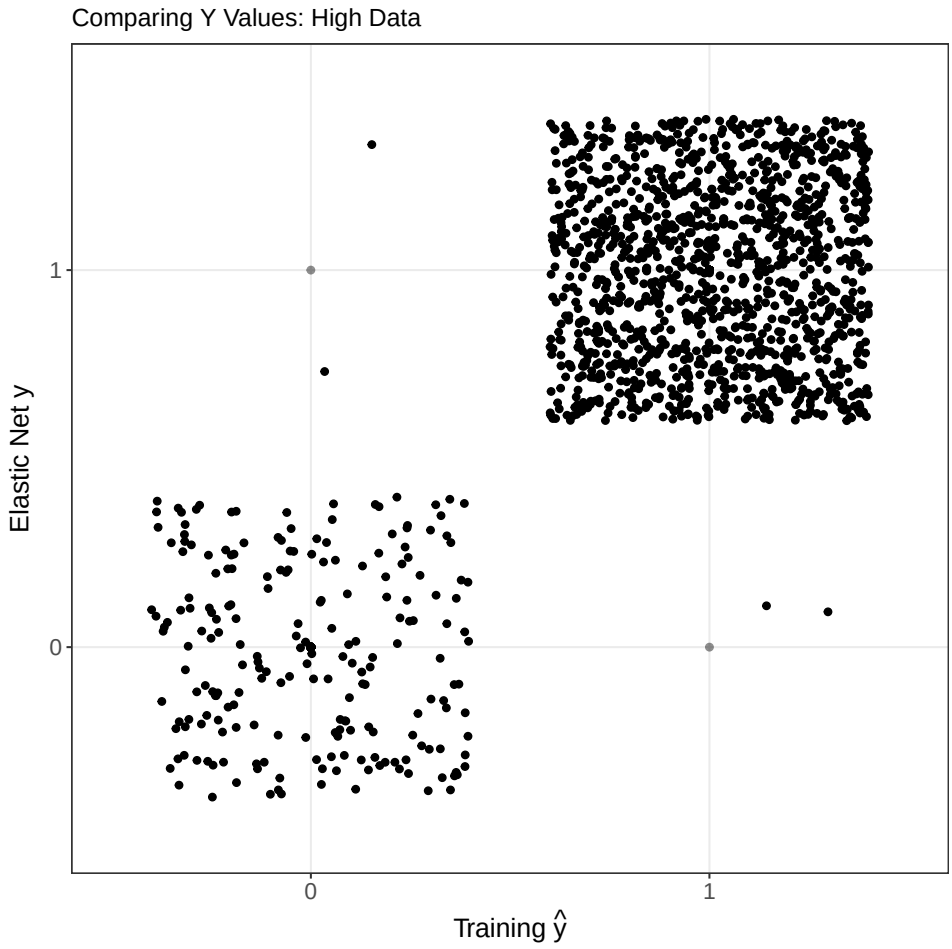
```
# Ballanced:
yhat = predict(enet.Bal)
qplot(train.data.1.Bal$Y, yhat, alpha=I(0.25))+
     geom_jitter()+
     xlab( expression(paste("Training ", hat(y))))+
     ylab( expression(paste("Elastic Net ", y)))+
     theme_bw()+
     theme(axis.text=element_text(size=12),
           axis.title=element_text(size=14))+
     ggtitle("Comparing Y Values: Ballanced Data")
```



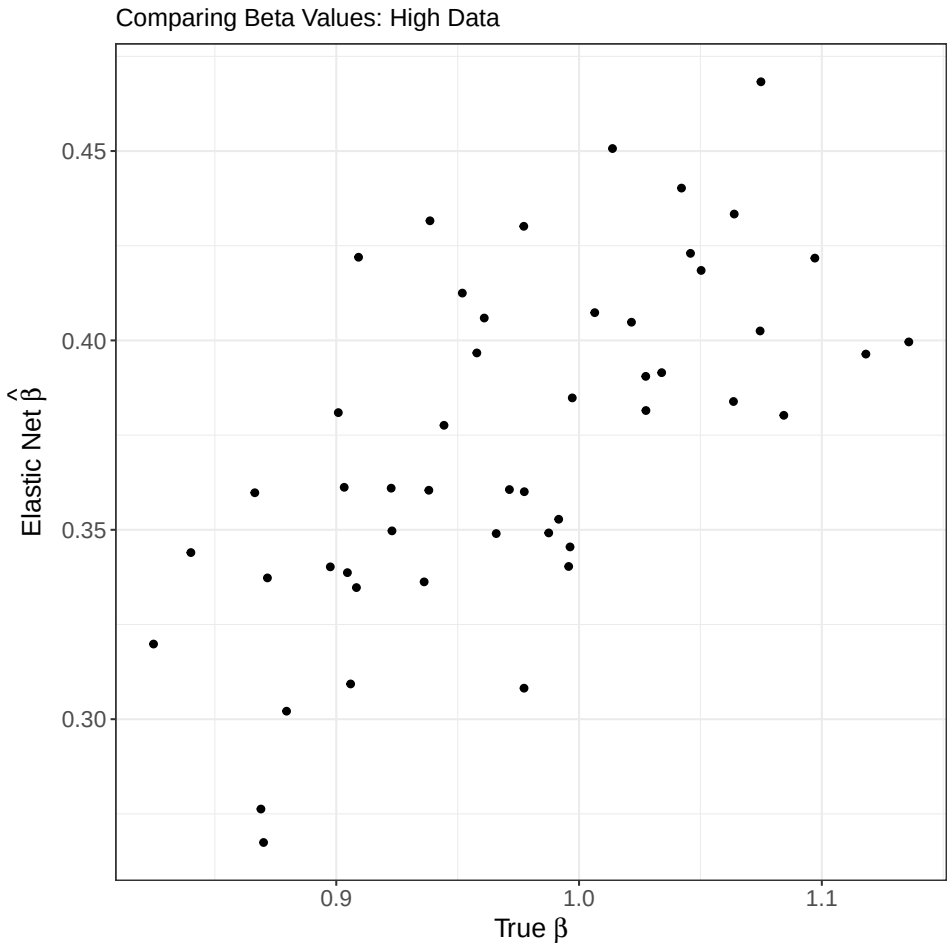
```
enet_beta = coef(enet.Bal$finalModel, enet.Bal$bestTune$lambda)
qplot(beta, enet_beta[-1])+
     xlab( expression(paste("True " , beta)))+
     ylab( expression(paste("Elastic Net " , hat(beta))))+
     theme_bw()+
     theme(axis.text=element_text(size=12),
           axis.title=element_text(size=14))+
     ggtitle("Comparing Beta Values: Ballanced Data")
```



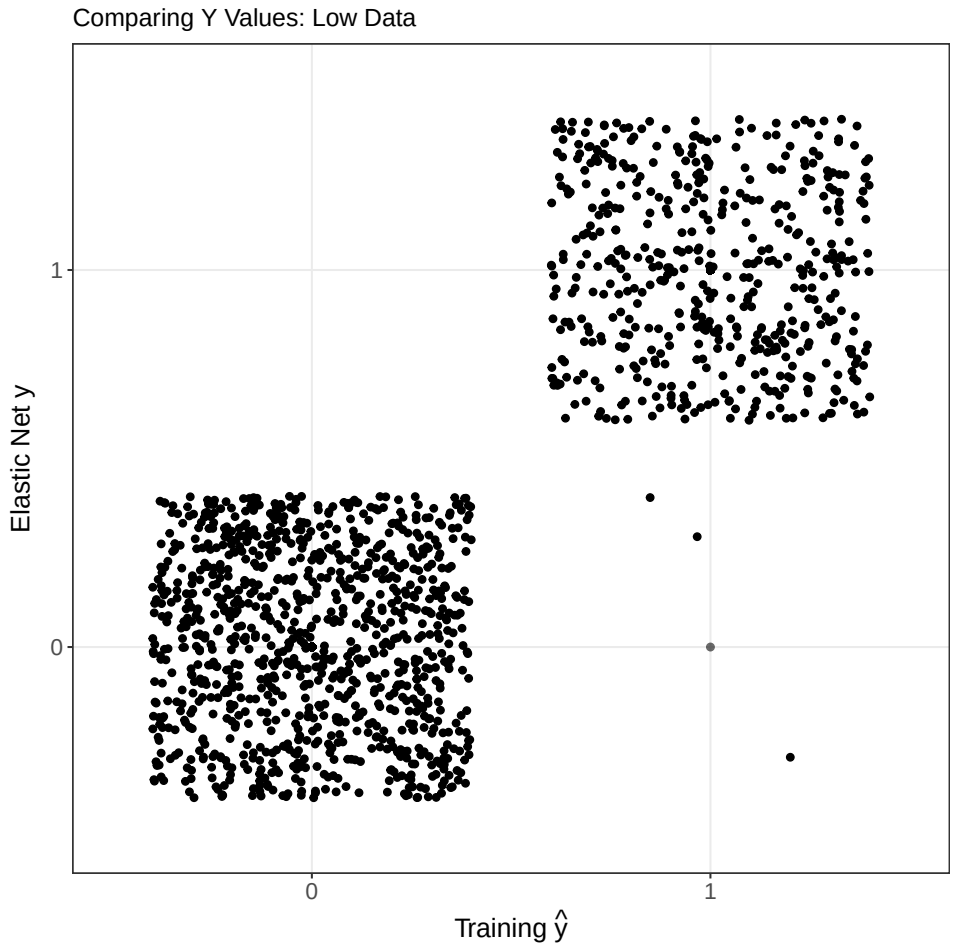

```
# High:
yhat = predict(enet.High)
qplot(train.data.2.High$Y2, yhat, alpha=I(0.25))+
     geom_jitter()+
     xlab( expression(paste("Training ", hat(y))))+
     ylab( expression(paste("Elastic Net ", y)))+
     theme_bw()+
     theme(axis.text=element_text(size=12),
           axis.title=element_text(size=14))+
     ggtitle("Comparing Y Values: High Data")
```



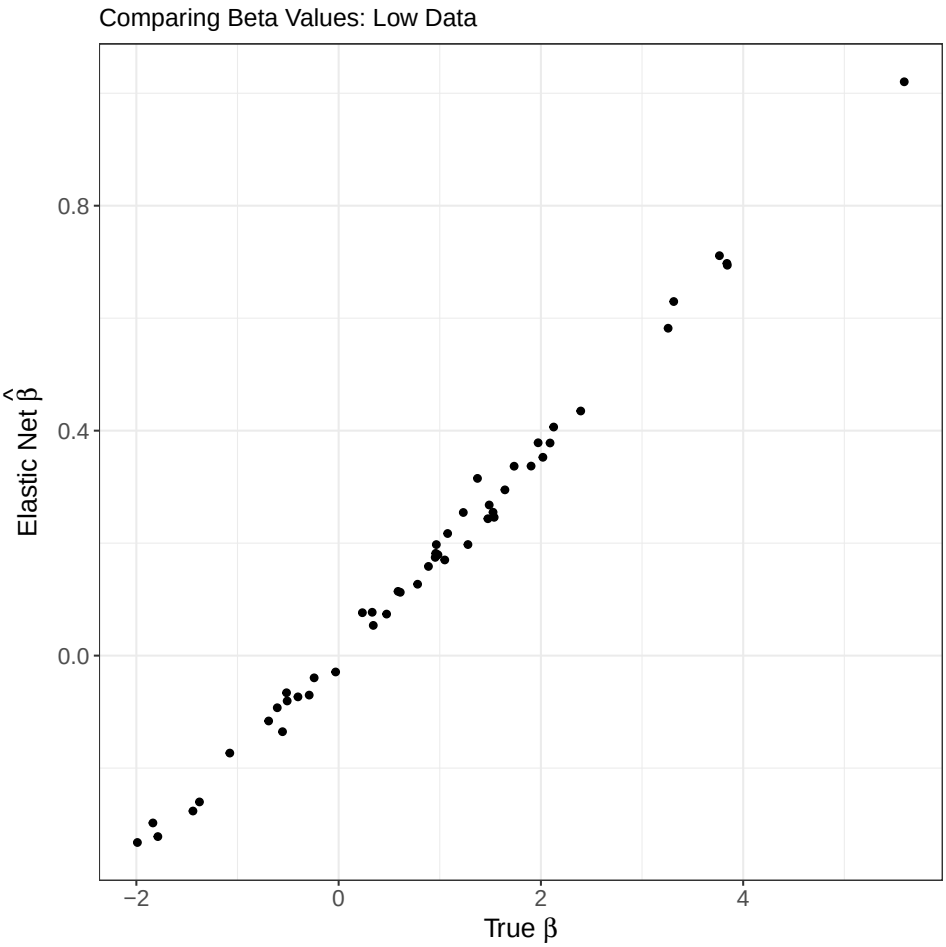

```
enet_beta = coef(enet.High$finalModel, enet.High$bestTune$lambda)
qplot(beta2, enet_beta[-1])+
     xlab( expression(paste("True " , beta)))+
     ylab( expression(paste("Elastic Net " , hat(beta))))+
     theme_bw()+
     theme(axis.text=element_text(size=12),
           axis.title=element_text(size=14))+
     ggtitle("Comparing Beta Values: High Data")
```




```
# Low:

yhat = predict(enet.Low)
qplot(train.data.3.Low$Y3, yhat, alpha=I(0.25))+
     geom_jitter()+
     xlab( expression(paste("Training ", hat(y))))+
     ylab( expression(paste("Elastic Net ", y)))+
     theme_bw()+
     theme(axis.text=element_text(size=12),
           axis.title=element_text(size=14))+
     ggtitle("Comparing Y Values: Low Data")
```




```
enet_beta = coef(enet.Low$finalModel, enet.Low$bestTune$lambda)
qplot(beta3, enet_beta[-1])+
     xlab( expression(paste("True " , beta)))+
     ylab( expression(paste("Elastic Net " , hat(beta))))+
     theme_bw()+
     theme(axis.text=element_text(size=12),
           axis.title=element_text(size=14))+
     ggtitle("Comparing Beta Values: Low Data")
```



From these graphs we can see that the variance in our binary predictions (jitter used to show variation in points) is very low. That is, we have very little mis-predictions between our elastic net and our DGPs. Perhaps interesting however, our elastic net for the highly probabilistic dataset does a rather poor job at accurately predicting our points. This may likely be because of the lack of variation in the chances that our predictions *fail*. However, in the dataset where we have no coverage of successful trials (dataset 3, Low) we see that our elastic net predicts our beta values better. This directly relates to how our betas were produced, specifically in the values we entered for their standard deviations. So for the data we generated (or indeed that would exist in the real world) I'll say that our ballanced data predictions did the best, all things considered. Another less theoretical way of thinking about this is to just report results and choose the model with the best accuracy, which also happens to be the ballanced data model.

```
## enet.Bal$results
enet.Bal$results[as.numeric(rownames(enet.Bal$bestTune)),]

##     alpha lambda  Accuracy     Kappa  AccuracySD    KappaSD
## 102   0.1      0 0.9759946 0.9516849 0.005702119 0.01151968

## enet.High$results
enet.High$results[as.numeric(rownames(enet.High$bestTune)),]
```

```
##    alpha lambda  Accuracy    Kappa  AccuracySD    KappaSD
## 62   0.1      0 0.9726667 0.887025 0.002828427 0.01388796

## enet.Low$results
enet.Low$results[as.numeric(rownames(enet.Low$bestTune)),]

##     alpha lambda Accuracy     Kappa  AccuracySD    KappaSD
## 142   0.1      0    0.968 0.9277209 0.007542472 0.01703277
```

Let's see if this ballanced model works well at predicting our testing data.

```
yhat.Bal <- predict(enet.Bal, newdata = test.data.1.Bal)
## table(test.data.1.Bal$Y, yhat.Bal)  # Check Results
accuracy <- 475/500 ## Probability correct / All observation
accuracy

## [1] 0.95
```

The accuracy of our training elastic net "worked" at predicting the witheld test data about 95% of the time. This is just slightly less than the predicitve accuracy from our elastic net model for the training data, this makes a good deal of sense considering our DGP was ballanced. I imagine an issue with imballanced data (High or Low) would be the general accuracy of a model considering classifications upon which we have little information or coverage of. What classifications could we have little information on? Perhaps an analysis of fringe political groups in relationship to more common classifications (Republican, Democrat). This would make it harder to predict classifications of these groups compared to a ballanced prediction between common groups.

Question 4:
One way of handling imbalanced training data is to force the sample into balance. You can do this by up-sampling the minority cases, or by down-sampling the majority cases. Let's first report the predictions we get on our test data from these imballanced data and see if up-sampling and down-sampling help us reach better predictions:

```
## High
yhat.High <- predict(enet.High, newdata = test.data.2.High)
## table(test.data.2.High$Y2, yhat.High)  # Check Results
accuracy  <- 492/500 ## Probability correct / All observation
accuracy

## [1] 0.984

## Low
yhat.Low <- predict(enet.Low, newdata = test.data.3.Low)
## table(test.data.3.Low$Y3, yhat.Low)  # Check Results
accuracy <- 490/500 ## Probability correct / All observation
accuracy
```

```
## [1] 0.98
```

We can see our data is incredibly accurate at predicting our test data, let's see if this goes up or down based up down-sampling or up-sampling our data.

```
### Up Sampling
# High Data
train.up.High <- upSample(x= train.data.2.High[, -ncol(train.data.2.High)],
                          y= train.data.2.High$Y2)
## table(train.up.high$Class) #Checking
# Low Data
train.up.Low  <- upSample(x= train.data.3.Low[, -ncol(train.data.3.Low)],
                          y= train.data.3.Low$Y3)
## table(train.up.low$Class)


### Down Sampling
# High Data
train.down.High <- downSample(x= train.data.2.High[, -ncol(train.data.2.High)],
                              y= train.data.2.High$Y2)
## table(train.down.High$Class)

# Low Data
train.down.Low <- downSample(x= train.data.3.Low[, -ncol(train.data.2.High)],
                             y= train.data.3.Low$Y3)
## table(train.down.Low$Class)
```

With these data that ballance our data by either up-sampling or down-sampling we can refit our elastic net penalized regression model, I'll save the vizualizations and report the results:

```
## Trained Up Data:
enet.up.high <- train(Class~., method="glmnet",
                      tuneGrid=expand.grid(alpha=seq(0, 1, 0.1),
                                           lambda=seq(0, 200, 1)),
                      data=train.up.High,
                      preProcess=c("center"),
                      trControl=trainControl(method="cv",number=2, search="grid"))
# plot(enet.up.high)
enet.up.low <- train(Class~., method="glmnet",
                     tuneGrid=expand.grid(alpha=seq(0, 1, 0.1),
                                          lambda=seq(0, 200, 1)),
                     data=train.up.Low,
                     preProcess=c("center"),
                     trControl=trainControl(method="cv",number=2, search="grid"))
# plot(enet.up.low)
```

```
## Trained Down Data:
enet.down.high <- train(Class~., method="glmnet",
                        tuneGrid=expand.grid(alpha=seq(0, 1, 0.1),
                                             lambda=seq(0, 200, 1)),
                        data=train.down.High,
                        preProcess=c("center"),
                        trControl=trainControl(method="cv",number=2, search="grid"))
# plot(enet.down.high)
enet.down.low <- train(Class~., method="glmnet",
                       tuneGrid=expand.grid(alpha=seq(0, 1, 0.1),
                                            lambda=seq(0, 200, 1)),
                       data=train.down.Low,
                       preProcess=c("center"),
                       trControl=trainControl(method="cv",number=2, search="grid"))
# plot(enet.down.low)
```

Here are the results for each of these, as before:

```
enet.up.high$results[as.numeric(rownames(enet.up.high$bestTune)),]

##      alpha lambda  Accuracy     Kappa AccuracySD     KappaSD
## 1006   0.5      0 0.9868115 0.9736233 0.00219428 0.004388066

enet.up.low$results[as.numeric(rownames(enet.up.low$bestTune)),]

##      alpha lambda  Accuracy     Kappa AccuracySD     KappaSD
## 1006   0.5      0 0.9879397 0.9758793 0.00284264 0.005684896

enet.down.high$results[as.numeric(rownames(enet.down.high$bestTune)),]

##   alpha lambda  Accuracy     Kappa AccuracySD    KappaSD
## 1     0      0 0.8933649 0.7867525 0.01005365 0.02003099

enet.down.low$results[as.numeric(rownames(enet.down.low$bestTune)),]

##     alpha lambda Accuracy     Kappa  AccuracySD     KappaSD
## 403   0.2      0 0.960392 0.9207839 0.002911342 0.005822684
```

And let's test these models back against our test data to see if they do better once up-scaled and down-scaled.

```
## up-sampling
yhat.up.high <- predict(enet.up.high, newdata = test.data.2.High)
#table(test.data.2.High$Y2, yhat.up.high)  # Check Results
```

```
accuracy  <- 493/500 ## Probability correct / All observation
accuracy

## [1] 0.986

yhat.up.low <- predict(enet.up.low, newdata = test.data.3.Low)
#table(test.data.3.Low$Y3, yhat.up.low)
accuracy  <- 486/500
accuracy

## [1] 0.972

## down-sampling
yhat.down.high <- predict(enet.down.high, newdata = test.data.2.High)
#table(test.data.2.High$Y2, yhat.down.high)
accuracy <- 475/500
accuracy

## [1] 0.95

yhat.down.low <- predict(enet.down.low, newdata = test.data.3.Low)
#table(test.data.3.Low$Y3, yhat.down.low)
accuracy <- 475/500
accuracy

## [1] 0.95
```

Our results for upsampling and downsampling do slightly worse than our predictions to begin with (around 98%). Why this might be is because the specification for our elastic nets that generated those highly accurate predictions were narrowed around lambda more rigourously than the elastic nets that were generated for these up-sampled and down-sampled data. The latter elastic nets were just steps of 1 for each lambda value from zero to 200. I'm certain if I had a bigger classification problem, or if my elastic nets were more rigorously combed to reach the "deepest" point in gradient descent we would have generated *slightly* more accurate predictions by up-sampling and down-sampling.

Question 5: (1)
Let's try out some support vector models (SVMs) to see if we can classify our data better than before:

```
# Some errors from masking, trying to prevent these here to little avail,
# nonetheless everything still works.
# detach(ggplot2)
library(kernlab)
```

```
# Linear SVM
linear.svm.bal  = train(Y~.,
                        data=train.data.1.Bal,
                        method="svmLinear")

linear.svm.bal$results$Accuracy

## [1] 0.9654228

# A polynomial kernal:
poly.svm.bal    = train(Y~.,
                        data=train.data.1.Bal,
                        method="svmPoly")

poly.svm.bal$results[as.numeric(rownames(poly.svm.bal$bestTune)),]

##   degree scale C  Accuracy    Kappa AccuracySD    KappaSD
## 9      1   0.1 1 0.9548612 0.908971 0.01016063 0.02054598

# Radial kernal:
radial.svm.bal  = train(Y~.,
                        data=train.data.1.Bal,
                        method="svmRadialCost")

radial.svm.bal$results[as.numeric(rownames(radial.svm.bal$bestTune)),]

##      C Accuracy Kappa AccuracySD KappaSD
## 1 0.25 0.537571     0 0.01947352       0
```

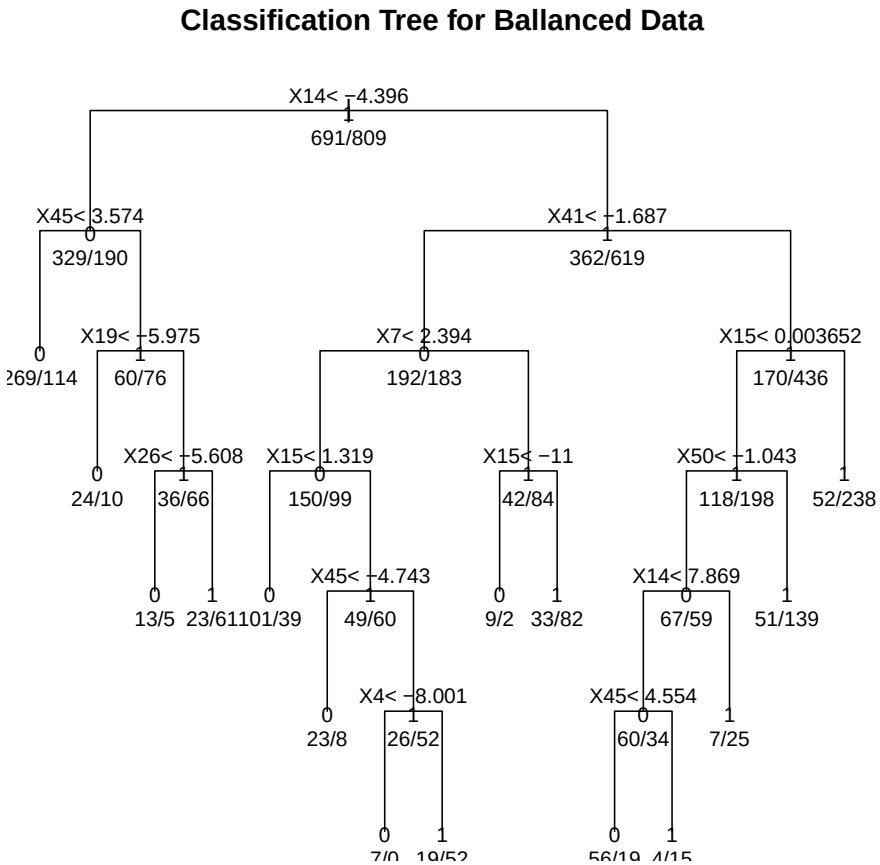
An SVM classifies data "better" by calculating the distances between points and a line (or plane) drawn through the data. The minimal errors between these points and the plane generates more reliable classification models. This is different from a strandard regression least-squared-errors approach because this uses Euclidean distance, which is different from a vertically measured residual. Beyond this, we can specify various kernals to attempt to classify our data in various dimensions. Overall our data seem to classify about the same as the standard elastic net for ballanced data (and who could argue with an extremely high classification accuracy anyways).
Let's see how these work on running our high and low data, specifically the linear SVM because it worked well in the ballanced data, and it's computationally simple:

```
### High and Low data linear SVMs:

# High
linear.svm.high  = train(Y2~.,
                         data=train.data.2.High,
```

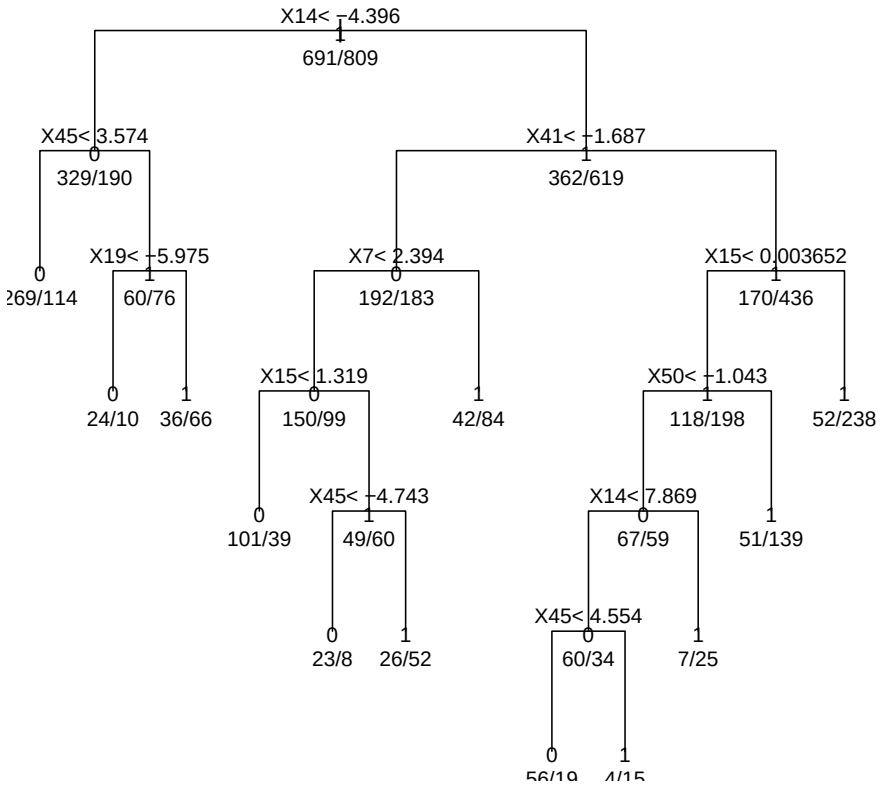
```
                        method="svmLinear")

linear.svm.high$results$Accuracy  # 97% accuracy

## [1] 0.9691195

# Low
linear.svm.low  = train(Y3~.,
                        data=train.data.3.Low,
                        method="svmLinear")

linear.svm.low$results$Accuracy  # 97% accuracy

## [1] 0.9679913
```

It seems as though these work a little bit better than our elastic net predictions! Let's try out regression trees as a way to further analize our ballanced data and low predictive data:

```
# Using "rpart", may need to reload R to get things to work.
library(rpart)

# grow tree
tree.bal <- rpart(Y~.,
             method="class", data=train.data.1.Bal)
## printcp(tree.bal) # display the results
## plotcp(tree.bal) # visualize cross-validation results
## summary(tree.bal) # detailed summary of splits
### These plots and data are useful for checking, let's generate vizualizations though;

# plot tree
plot(tree.bal, uniform=TRUE,
     main="Classification Tree for Ballanced Data")
text(tree.bal, use.n=TRUE, all=TRUE, cex=.8)
```

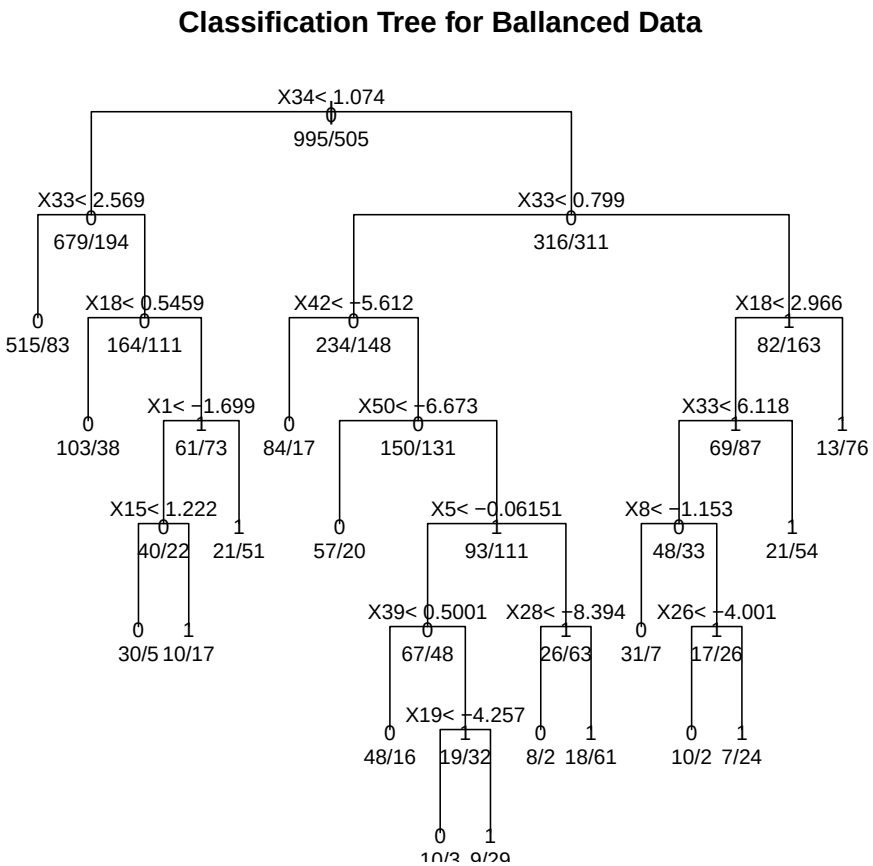



```
# create attractive postscript plot of tree
post(tree.bal,
     title = "Classification Tree for Ballanced Data")

# then we have a lot of options off of this, including pruning the tree back,
# or making -multiple- trees into a forrest and
# selecting the best one! (intutively a form of advanced bootstrapping!)

# prune the tree, by the error that was previously generated above
ptree.bal <- prune(tree.bal,
                   cp=tree.bal$cptable[which.min(tree.bal$cptable[,"xerror"]),"CP"])

# plot the pruned tree
plot(ptree.bal, uniform=TRUE,
     main ="Pruned Classification Tree for Ballanced Data")
text(ptree.bal, use.n=TRUE, all=TRUE, cex=.8)
```

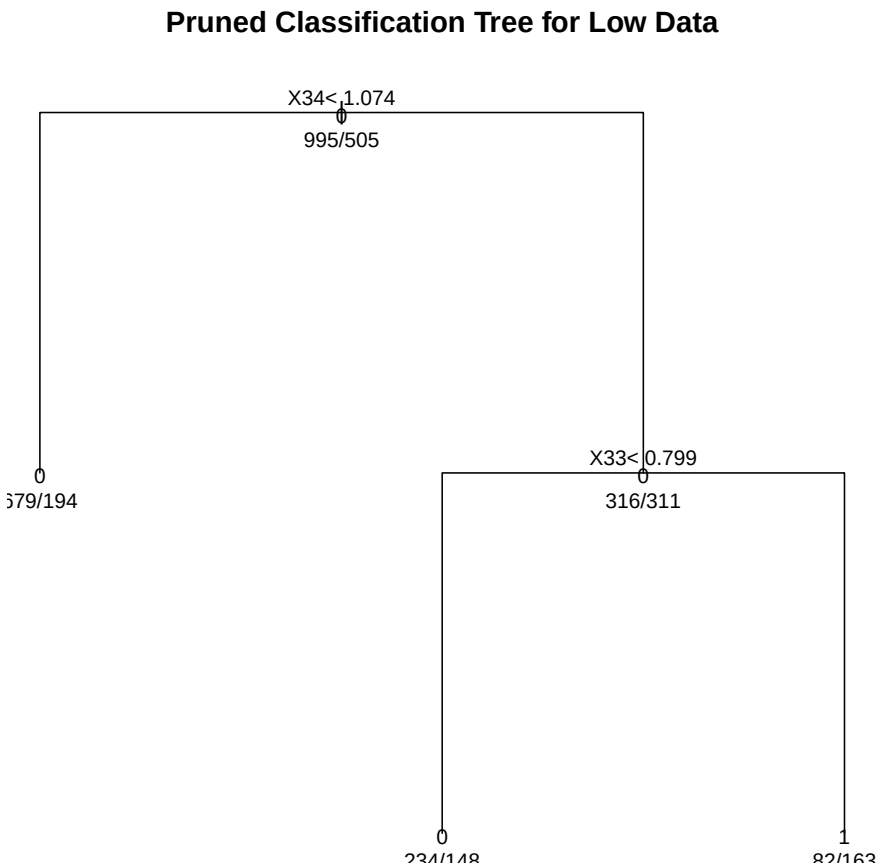



```
post(ptree.bal,
     title = "Pruned Classification Tree for Ballanced Data")

# Great! But how does this work on our low predictive model, for example?

## Low Data:
# grow tree
tree.low <- rpart(Y3~.,
                  method="class", data=train.data.3.Low)

## printcp(tree.low) # display the results
## plotcp(tree.low)  # visualize cross-validation results
## summary(tree.low) # detailed summary of splits

# plot tree for the low data:
plot(tree.low, uniform=TRUE,
```

```
     main="Classification Tree for Ballanced Data")
text(tree.low, use.n=TRUE, all=TRUE, cex=.8)
```




```
# create attractive postscript plot of tree
post(tree.low,
     title = "Classification Tree for Low Data")

# prune the tree, by the error that was previously generated above
ptree.low <- prune(tree.low,
                   cp=tree.low$cptable[which.min(tree.low$cptable[,"xerror"]),"CP"])

# plot the pruned tree
plot(ptree.low, uniform=TRUE,
     main ="Pruned Classification Tree for Low Data")
text(ptree.low, use.n=TRUE, all=TRUE, cex=.8)
```




```
post(ptree.low,
     title = "Pruned Classification Tree for Low Data")
```

Overall, regression trees (here they are classification trees) can be used to see which predictors lead our classifications, and similar to how elastic nets will bring some predictors to zero we can "prune" these observations and vizualize this process better. We see that in our low predictive data we have so many observations that have high error that our tree becomes heavily pruned. Reporting these over other methods may be more intuitive for readers and reviewers.