wine_exploration

Chris DiSerafino

4/14/2020

R Markdown

Research question: how well can we classify a wine as red or white using neural networks? Does the number of hidden units or layers matter? How does starting values or scaling effect our results? The risk function of neural networks is not convex. Which method of finding a global minima is most effective in this instance?

Explore Data

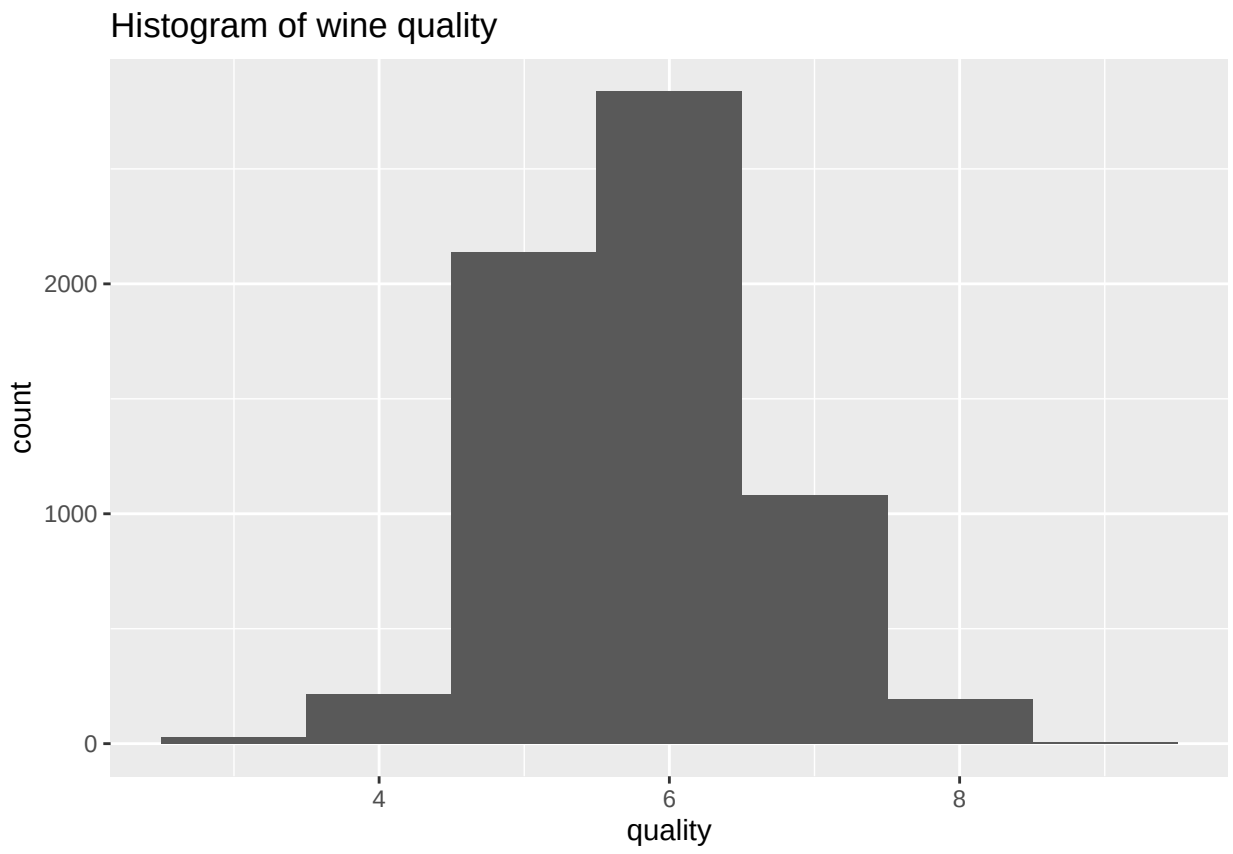
```
color_wines = color_wines[,2:13]
color_predict = color_wines[,1:11]
color_response = color_wines[,12]

quality_wines = quality_wines[,2:13]
quality_predict = quality_wines[,1:11]
quality_response = quality_wines[,12]

wines = wines[,2:14]

set.seed(13)
head(quality_wines)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.0             0.27        0.36           20.7     0.045
## 2           6.3             0.30        0.34            1.6     0.049
## 3           8.1             0.28        0.40            6.9     0.050
## 4           7.2             0.23        0.32            8.5     0.058
## 5           7.2             0.23        0.32            8.5     0.058
## 6           8.1             0.28        0.40            6.9     0.050
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  45                  170  1.0010 3.00      0.45     8.8
## 2                  14                  132  0.9940 3.30      0.49     9.5
## 3                  30                   97  0.9951 3.26      0.44    10.1
## 4                  47                  186  0.9956 3.19      0.40     9.9
## 5                  47                  186  0.9956 3.19      0.40     9.9
## 6                  30                   97  0.9951 3.26      0.44    10.1
##   quality
## 1       6
## 2       6
## 3       6
## 4       6
## 5       6
## 6       6
```

```r
head(color_wines)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.0             0.27        0.36           20.7     0.045
## 2           6.3             0.30        0.34            1.6     0.049
## 3           8.1             0.28        0.40            6.9     0.050
## 4           7.2             0.23        0.32            8.5     0.058
## 5           7.2             0.23        0.32            8.5     0.058
## 6           8.1             0.28        0.40            6.9     0.050
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol type
## 1                  45                  170  1.0010 3.00      0.45     8.8    0
## 2                  14                  132  0.9940 3.30      0.49     9.5    0
## 3                  30                   97  0.9951 3.26      0.44    10.1    0
## 4                  47                  186  0.9956 3.19      0.40     9.9    0
## 5                  47                  186  0.9956 3.19      0.40     9.9    0
## 6                  30                   97  0.9951 3.26      0.44    10.1    0
```

```r
#look at distribution of wine quality
ggplot(wines) +
  geom_histogram(aes(x = quality), binwidth = 1) +
  ggtitle("Histogram of wine quality")
```

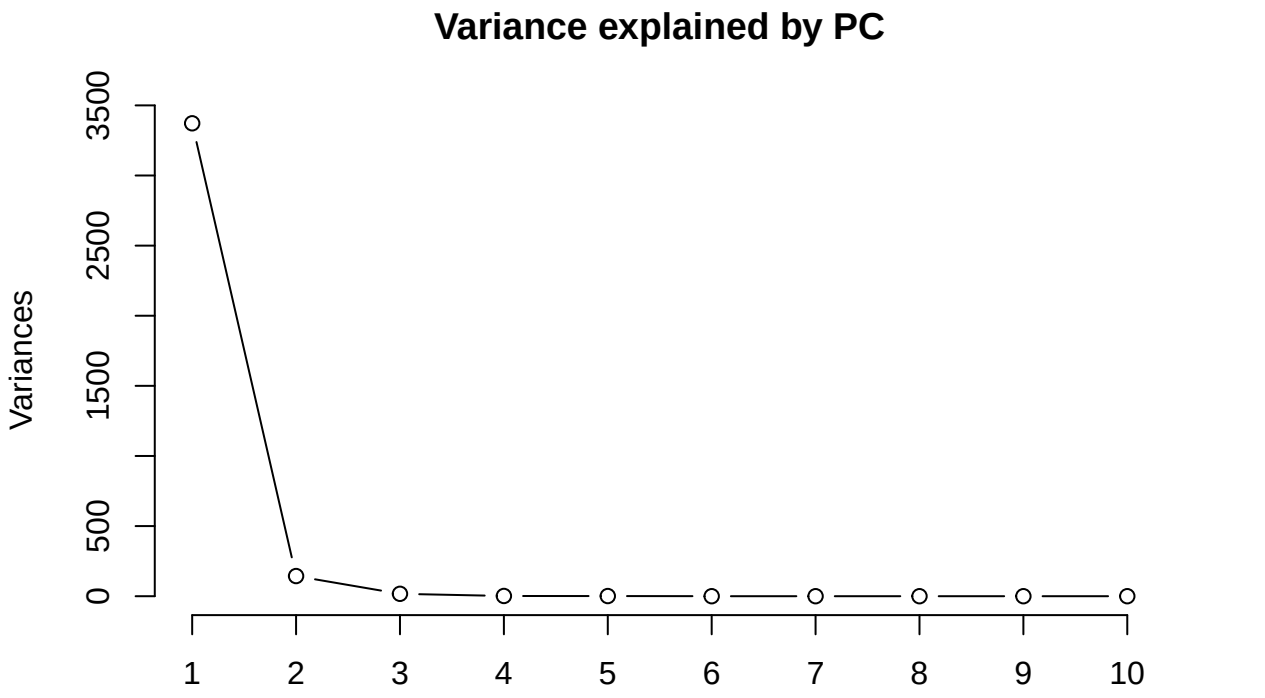



```r
#how would we want to classify these wines?


#Run a PCA
pcs = prcomp(color_predict)
```

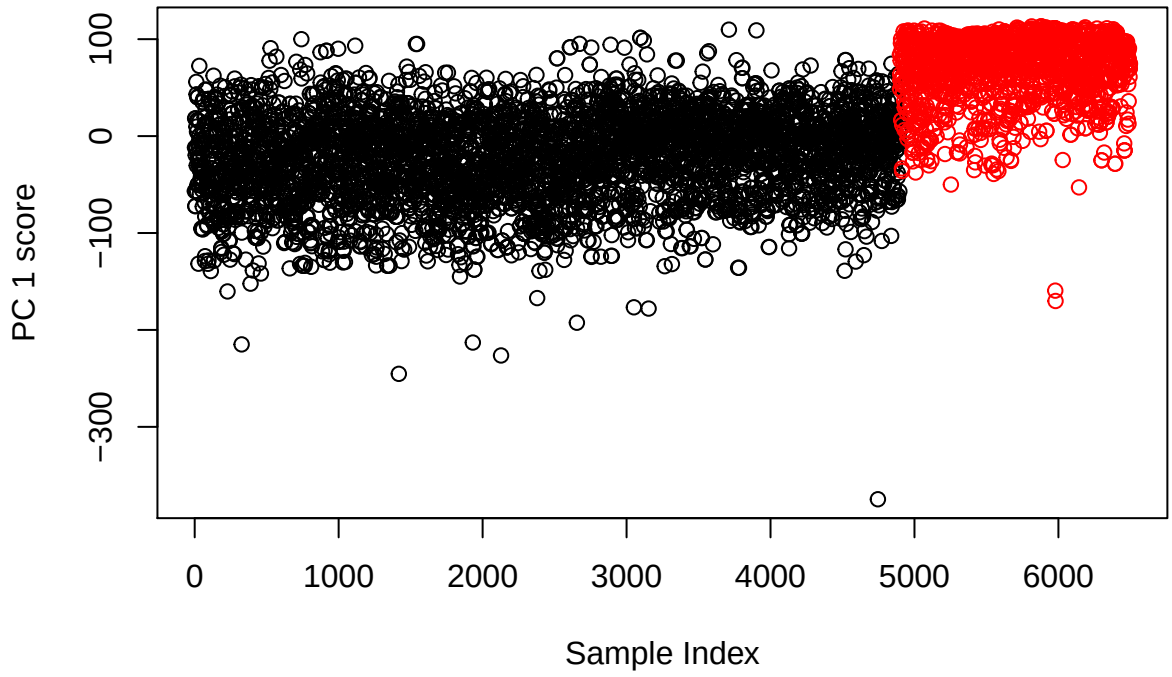
```r
#Summarize the pcs
summary(pcs)
```

```
## Importance of components:
##                             PC1      PC2     PC3     PC4    PC5     PC6     PC7
## Standard deviation     58.0698 11.98513 4.13082 1.28066 1.0328 0.17714 0.14464
## Proportion of Variance  0.9538  0.04063 0.00483 0.00046 0.0003 0.00001 0.00001
## Cumulative Proportion   0.9538  0.99439 0.99921 0.99968 1.0000 0.99999 0.99999
##                            PC8    PC9    PC10      PC11
## Standard deviation     0.1211 0.1031 0.02787 0.0007517
## Proportion of Variance 0.0000 0.0000 0.00000 0.0000000
## Cumulative Proportion  1.0000 1.0000 1.00000 1.0000000
```

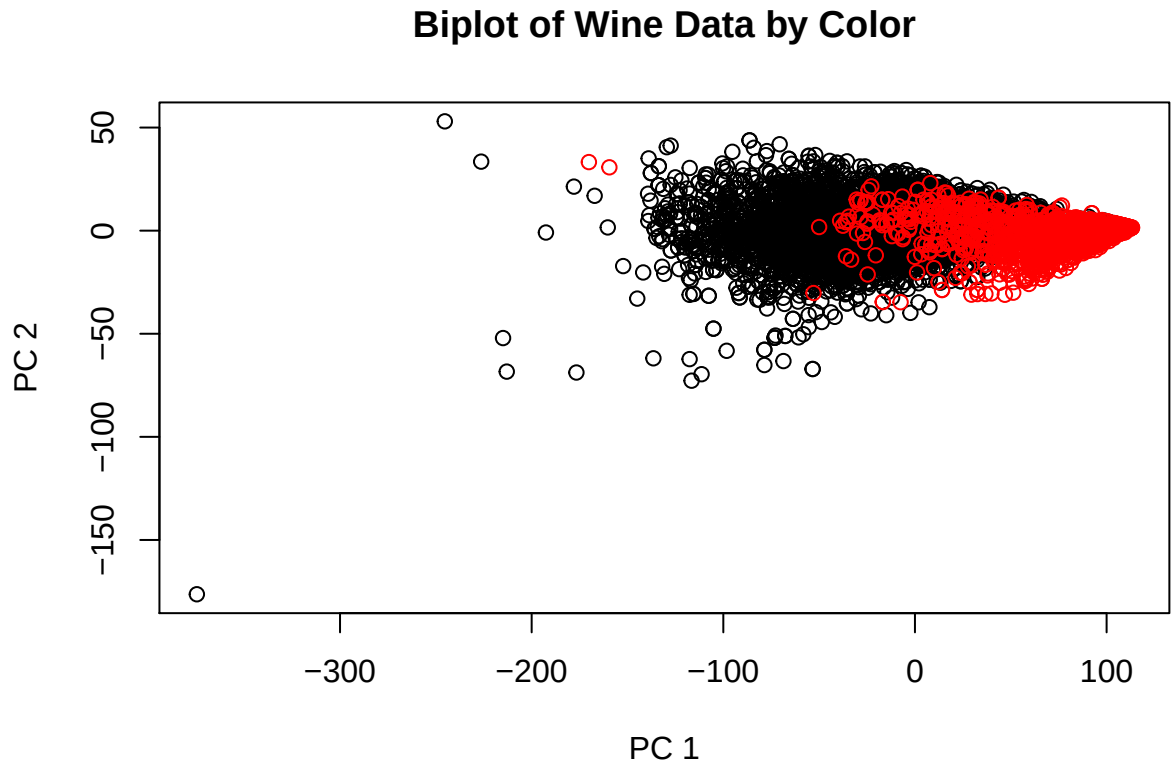
```r
screeplot(pcs, type = "lines", main = "Variance explained by PC")
```



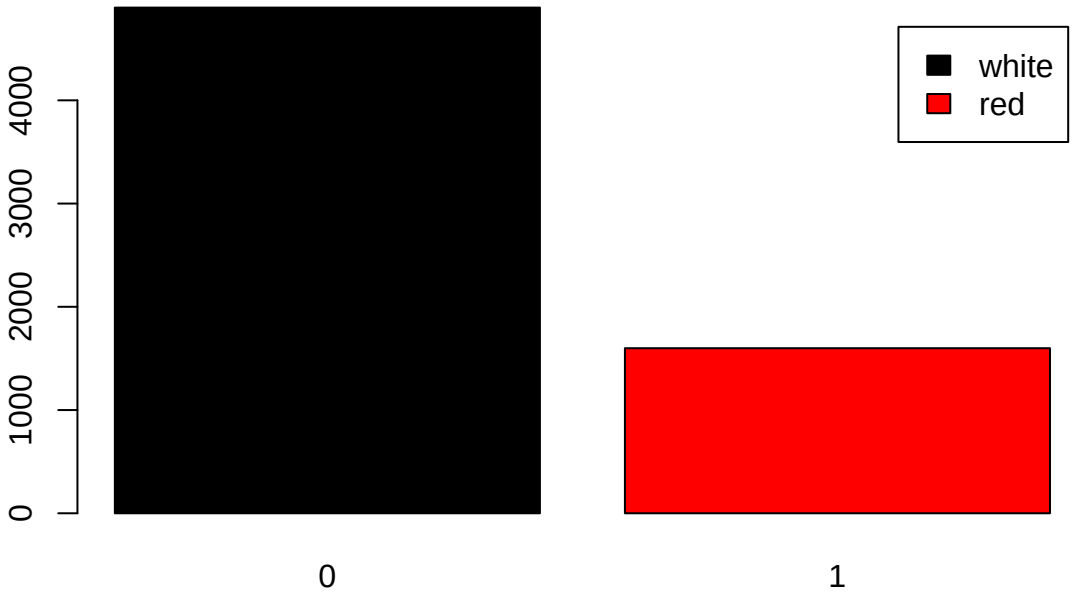

```r
plot(pcs$x[,1],
      main = paste("PC", eval(1)),
      xlab = "Sample Index",
      ylab = paste("PC", eval(1), "score"),
      col = color_response + 1)
```



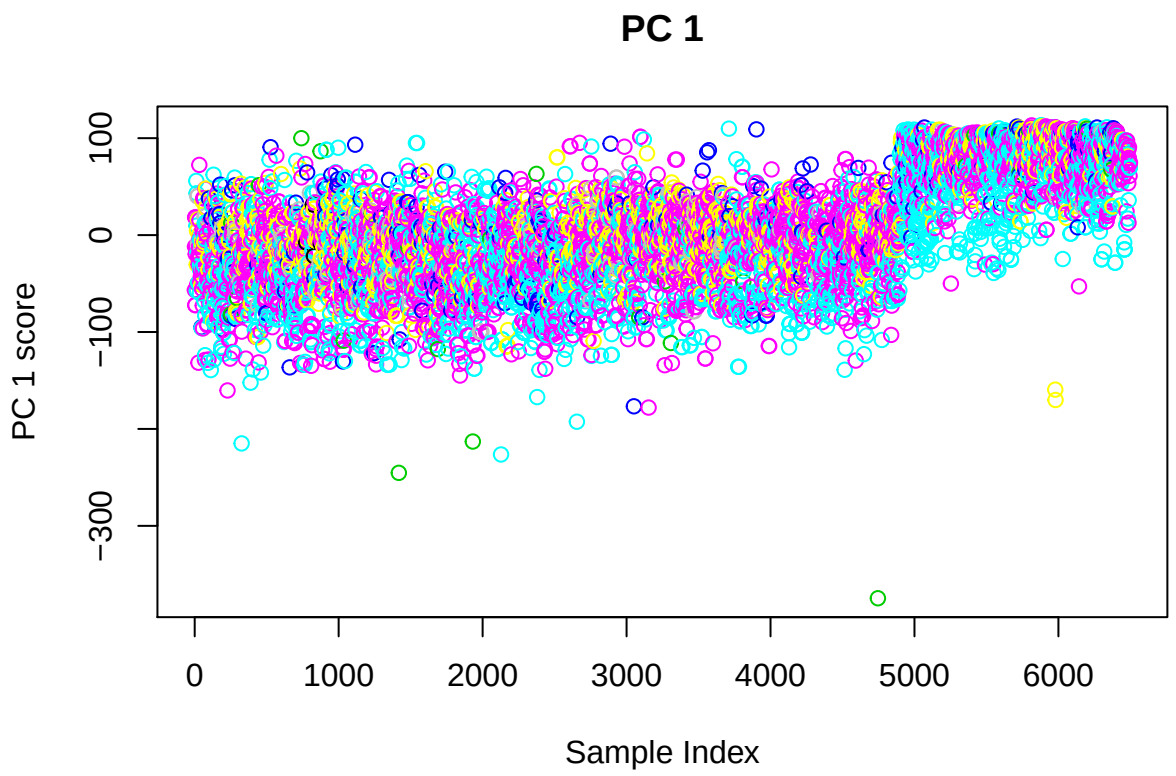

```
#first pc seems to do most of work in seperating by wine color

plot(pcs$x[,1:2],
     main = "Biplot of Wine Data by Color",
     xlab = "PC 1", ylab = "PC 2",
     col = color_response + 1)
```



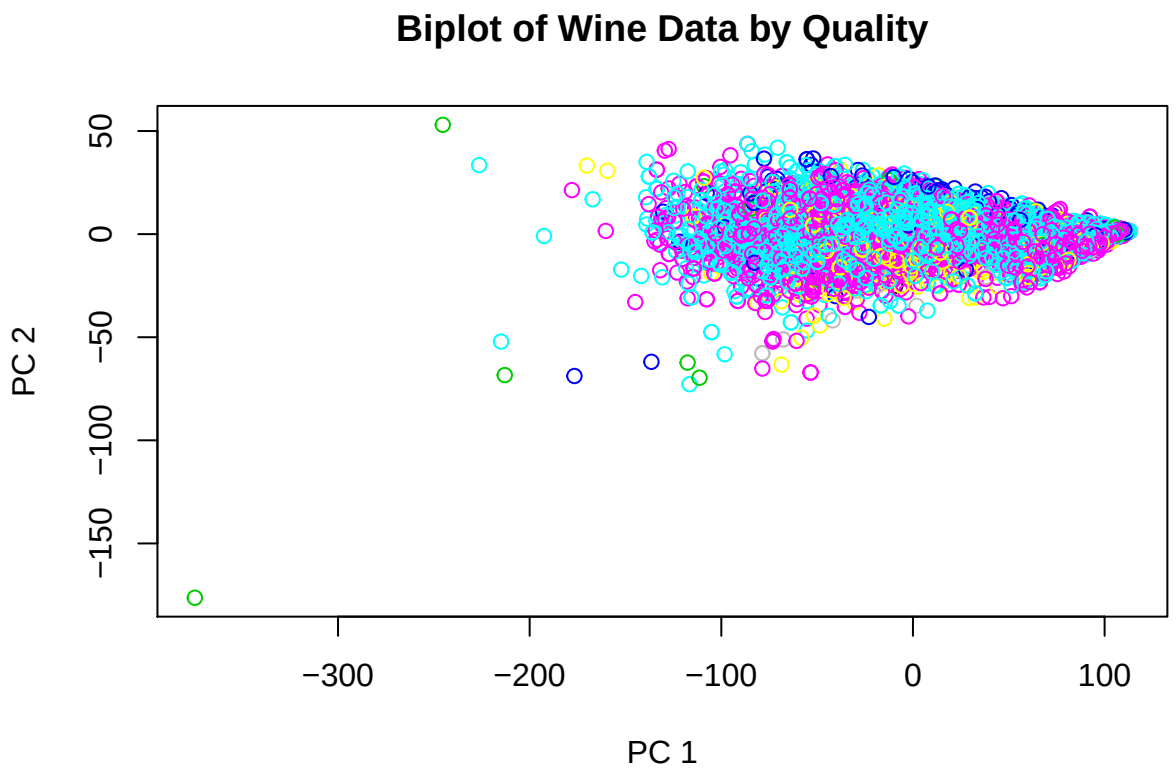

```
barplot(table(color_response), col = unique(color_response + 1), legend.text = c("white", "red"))
```



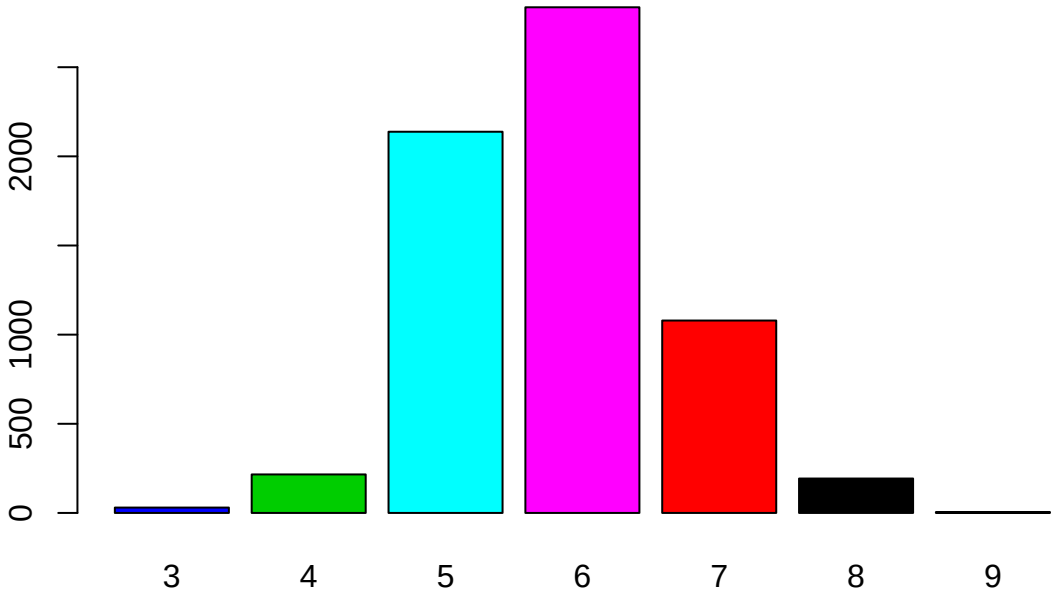

```
plot(pcs$x[,1],
     main = paste("PC", eval(1)),
     xlab = "Sample Index",
     ylab = paste("PC", eval(1), "score"),
     col = quality_response)
```



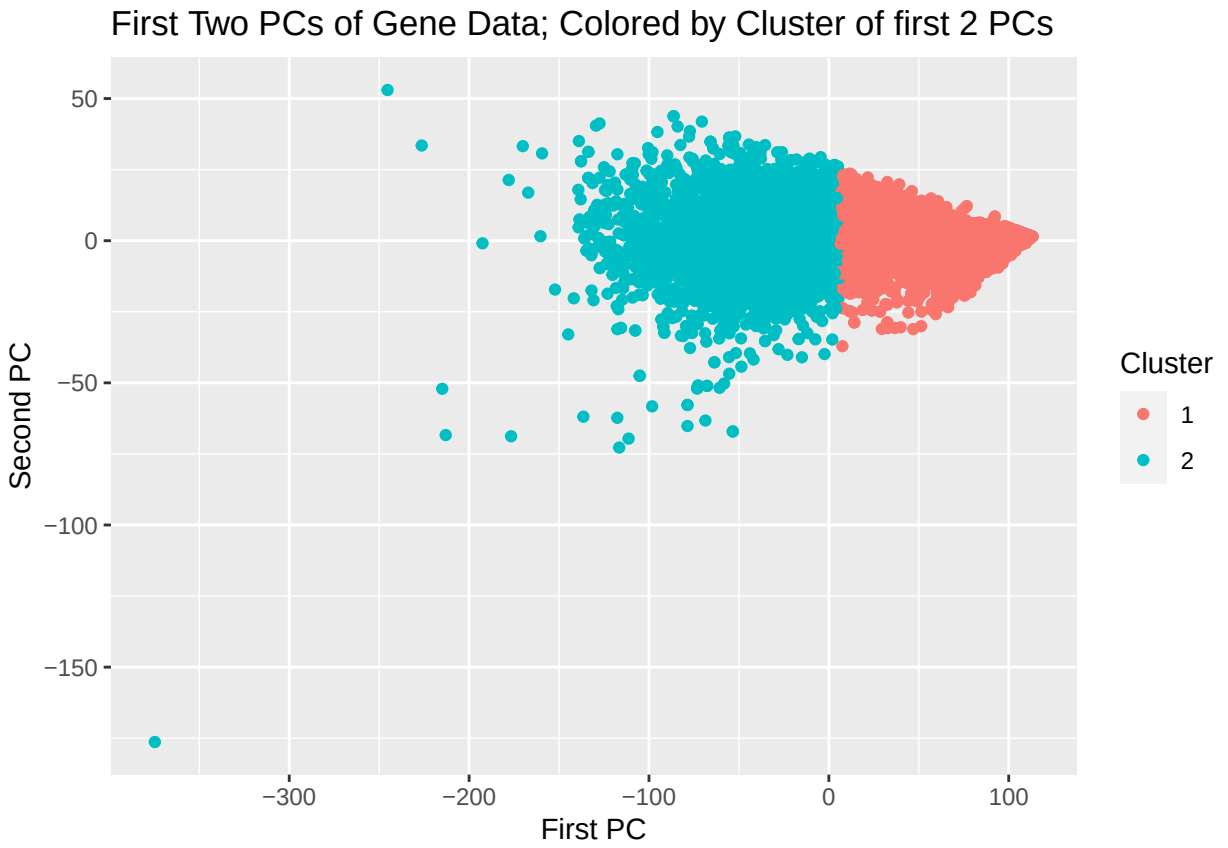

```
plot(pcs$x[,1:2],
     main = "Biplot of Wine Data by Quality",
     xlab = "PC 1", ylab = "PC 2",
     col = quality_response)
```



```r
barplot(table(quality_response), col = unique(quality_response - 2))
```




```r
#only show first two pcs because they account for most of variance
k.pca = kmeans(pcs$x[,1:2], centers = 2)
cluster_update = as.factor(k.pca$cluster)
x <- pcs$x
ggplot() +
  geom_point(aes(x = x[,1], y = x[,2], col = cluster_update))+
  xlab("First PC") +
  ylab("Second PC") +
  scale_colour_discrete(name = "Cluster") +
  ggtitle("First Two PCs of Gene Data; Colored by Cluster of first 2 PCs")
```




```
#clustering using kmeans with 2 centers seems to be a good approximation of color
```

Analysis: One PC explains over 95 percent of the variance, and two PCs explain over 99 percent of the variance! This is something we should definitely mention, we can plot to two dimensions and still have most of the variance accounted for

Create Neural Network

```
set.seed(13)
#NN to predict color
#min max normalization
max = apply(color_wines, 2 , max)
min = apply(color_wines, 2 , min)
color_wines = as.data.frame(scale(color_wines, center = min, scale = max - min))

#divide into training set and testing set
training_size = round(.75 * nrow(color_wines))
indices = sample(1:nrow(color_wines), training_size)
training_set = color_wines[indices,]
testing_set = color_wines[-(indices),]

NN = neuralnet(type ~ citric.acid + alcohol + chlorides, training_set, hidden = 3 , linear.output = F)

#simpler model
plot(NN)

NN = neuralnet(type ~ ., training_set, hidden = 3 , linear.output = F)
```

```r
# plot neural network
plot(NN)
```

```r
predict_testNN = compute(NN, testing_set[,c(1:12)])
predict_testNN = predict_testNN$net.result

predicted_labels = c();
predicted_labels = (predict_testNN[,1] >= 0.5) *1

# Calculate test Risk
nn_risk = sum(testing_set$type == predicted_labels)/length(predicted_labels)
if (nn_risk >= 0.5) {
  nn_risk = 1 - nn_risk
}
nn_risk_test = nn_risk

predict_trainNN = compute(NN, training_set[,c(1:12)])
predict_trainNN = predict_trainNN$net.result

predicted_labels = c();
predicted_labels = (predict_trainNN[,1] >= 0.5) *1

# Calculate train Risk
nn_risk = sum(training_set$type == predicted_labels)/length(predicted_labels)
if (nn_risk >= 0.5) {
  nn_risk = 1 - nn_risk
}
nn_risk_train = nn_risk

model = c(1, 1)
error = c(nn_risk_train, nn_risk_test)
data = c("train", "test")
total_error = data.frame(cbind(model, error, data))

#plot risk
ggplot(total_error) +
  geom_col(aes( x = model, y = error, color = data), position = position_dodge()) +
  ggtitle("Testing and Training Error Rate of Model for Wine Type")
```


```
#Neural Network to predict quality
#good and poor quality divide
quality_wines$good = NA
quality_wines$good[quality_wines[,12] > 5] = 1
quality_wines$good[quality_wines[,12] <= 5] = 0
quality_wines = quality_wines[,-(12)]

#min max normalization
set.seed(14)
max = apply(quality_wines, 2 , max)
min = apply(quality_wines, 2 , min)
scaled_wines = as.data.frame(scale(quality_wines, center = min, scale = max - min))

#divide into training set and testing set
training_size = round(.75 * nrow(scaled_wines))
indices = sample(1:nrow(scaled_wines), training_size)
training_set = scaled_wines[indices,]
testing_set = scaled_wines[-(indices),]



#NN.quality = neuralnet(good ~ ., training_set, hidden = 3, act.fct = "tanh", linear.output = FALSE)

#this function is very inconsistent, I believe the tanh activation
#function is creating errors in back propogation for this dataset
#prevents document from knitting so I am replacing with another
#neural network
```

```
NN.quality = neuralnet(good ~ ., training_set, hidden = 2, linear.output = FALSE)

NN.lin = neuralnet(good ~ ., training_set, hidden = 3, linear.output = FALSE)

# plot neural network
#plot(NN.quality)

predict_test = compute(NN.quality, testing_set[,c(1:12)])
predict_lin = compute(NN.lin, testing_set[,c(1:12)])
#predict_test = (predict_test$net.result * (max(quality_wines$quality) - min(quality_wines$quality))) +
#testing_quality = as.factor(quality_wines[-(indices),"quality"])


#for tclassification
predicted_labels = (predict_test$net.result[,1] >= 0.5) *1
predicted_labels_lin = (predict_lin$net.result[,1] >= 0.5) *1

# Calculate Test Risk for both models
nn_risk = sum(testing_set$good == predicted_labels)/length(predicted_labels)
if (nn_risk >= 0.5) {
  nn_risk = 1 - nn_risk
}
nn_risk_test = nn_risk

nn_risk2 = sum(testing_set$good == predicted_labels_lin)/length(predicted_labels_lin)
if (nn_risk2 >= 0.5) {
  nn_risk2 = 1 - nn_risk2
}
nn_risk_test2 = nn_risk2

predict_train = compute(NN.quality, training_set[,c(1:12)])
predict_train = predict_train$net.result

predict_train_lin = compute(NN.lin, training_set[,c(1:12)])
predict_train_lin = predict_train_lin$net.result

predicted_labels = c();
predicted_labels = (predict_train[,1] >= 0.5) *1

# Calculate Train Risk for both models
nn_risk = sum(training_set$good == predicted_labels)/length(predicted_labels)
if (nn_risk >= 0.5) {
  nn_risk = 1 - nn_risk
}
nn_risk_train = nn_risk

predicted_labels_lin = (predict_train_lin[,1] >= 0.5) *1

# Calculate Risk
nn_risk2 = sum(training_set$good == predicted_labels_lin)/length(predicted_labels_lin)
if (nn_risk2 >= 0.5) {
  nn_risk2 = 1 - nn_risk2
```

```
}
nn_risk_train2 = nn_risk2

model = c(1, 1, 2, 2)
error = c(nn_risk_train, nn_risk_test, nn_risk_train2, nn_risk_test2)
data = c("train", "test", "train", "test")
total_error = data.frame(cbind(model, error, data))

#plot errors
ggplot(total_error) +
  geom_col(aes(x = model, y = error, color = data), position = position_dodge()) +
  ggtitle("Testing and Training Error Rate of Models")
```