Computer Vision System - Dry Beans

David Pershall

9/21/2021

Introduction

This data set is provided by the UCI Machine Learning Repository at the following web address.

https://archive.ics.uci.edu/ml/datasets/Dry+Bean+Dataset

The data set summary is quoted below. Let's have a quick read...

"A computer vision system was developed to distinguish seven different registered varieties of dry beans with similar features in order to obtain uniform seed classification. For the classification model, images of 13,611 grains of 7 different registered dry beans were taken with a high-resolution camera. Bean images obtained by the computer vision system were subjected to segmentation and feature extraction stages, and a total of 16 features; 12 dimensions and 4 shape forms, were obtained from the grains."

This goal of this project is to use the knowledge I have gained during HarvardX's Data Science Professional Certificate program to compose several multi-class classification algorithms in an attempt to match or beat the accuracy reported by Koklu and Oskan in their paper titled "Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques". They used a support vector machine to achieve a 93.13% accuracy rate. You can read the abstract of their paper online at ScienceDirect.com by following this link.

https://www.sciencedirect.com/science/article/abs/pii/S0168169919311573?via%3Dihub

I have only read the abstract and not the paper itself in order to preserve my own creative responses in solving this multi-class classification problem.

Important Side Note An important note. Do not simply run all the chunks in this project without reading. I used parallel cloud computing for several of my models which could crash R if you run them locally on a system that doesn't meet certain requirements. I have provided detailed notes about the time and system requirements for each of those models as we approach them.

OK! Let's dive in and take a look.

Libraries

The following libraries were used for this project.

```
library(tidyverse)
library(caret)
library(readxl)
library(randomForest)
library(matrixStats)
library(GGally)
library(doParallel)
```

Pulling the data

Here we pull the data and take a look for any missing values.

```
tmp <- tempfile()
download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/00602/DryBeanDataset.zip", tmp

dat <- read_excel(unzip(tmp, "DryBeanDataset/Dry_Bean_Dataset.xlsx"))

glimpse(dat)
```

```
## Rows: 13,611
## Columns: 17
## $ Area            <dbl> 28395, 28734, 29380, 30008, 30140, 30279, 30477, 30519~
## $ Perimeter       <dbl> 610.291, 638.018, 624.110, 645.884, 620.134, 634.927, ~
## $ MajorAxisLength <dbl> 208.1781, 200.5248, 212.8261, 210.5580, 201.8479, 212.~
## $ MinorAxisLength <dbl> 173.8887, 182.7344, 175.9311, 182.5165, 190.2793, 181.~
## $ AspectRation    <dbl> 1.197191, 1.097356, 1.209713, 1.153638, 1.060798, 1.17~
## $ Eccentricity    <dbl> 0.5498122, 0.4117853, 0.5627273, 0.4986160, 0.3336797,~
## $ ConvexArea      <dbl> 28715, 29172, 29690, 30724, 30417, 30600, 30970, 30847~
## $ EquivDiameter   <dbl> 190.1411, 191.2728, 193.4109, 195.4671, 195.8965, 196.~
## $ Extent          <dbl> 0.7639225, 0.7839681, 0.7781132, 0.7826813, 0.7730980,~
## $ Solidity        <dbl> 0.9888560, 0.9849856, 0.9895588, 0.9766957, 0.9908933,~
## $ roundness       <dbl> 0.9580271, 0.8870336, 0.9478495, 0.9039364, 0.9848771,~
## $ Compactness     <dbl> 0.9133578, 0.9538608, 0.9087742, 0.9283288, 0.9705155,~
## $ ShapeFactor1    <dbl> 0.007331506, 0.006978659, 0.007243912, 0.007016729, 0.~
## $ ShapeFactor2    <dbl> 0.003147289, 0.003563624, 0.003047733, 0.003214562, 0.~
## $ ShapeFactor3    <dbl> 0.8342224, 0.9098505, 0.8258706, 0.8617944, 0.9419004,~
## $ ShapeFactor4    <dbl> 0.9987239, 0.9984303, 0.9990661, 0.9941988, 0.9991661,~
## $ Class           <chr> "SEKER", "SEKER", "SEKER", "SEKER", "SEKER", "SEKER", ~
```

```
anyNA(dat)
```

```
## [1] FALSE
```

Since there are no missing variables, let's continue forward with some analysis.

Pre-Processing and Exploritory Data Analysis

Let's create a new object to hold our data that will make it easier to scale/center.

```
db <- with(dat, list(x =as.matrix(dat[-17]), y = as.factor(dat$Class)))

class(db$x)
```

```
## [1] "matrix" "array"
```

```
class(db$y)
```

```
## [1] "factor"
```

```
# double checking that the observations were maintained
dim(db$x)[1]
```

```
## [1] 13611
```

```
dim(db$x)[2]
```

```
## [1] 16
```

Now that that is accomplished, I want to know how the classes of y (the types of beans) are represented on average in the data set?

```
mean(db$y == "BOMBAY")
```

```
## [1] 0.03835133
```

```
mean(db$y == "SEKER")
```

```
## [1] 0.1489237
```

```
mean(db$y == "CALI")
```

```
## [1] 0.1197561
```

```
mean(db$y == "DERMASON")
```

```
## [1] 0.2605246
```

```
mean(db$y == "HOROZ")
```

```
## [1] 0.1416501
```

```
mean(db$y == "SIRA")
```

```
## [1] 0.1936669
```

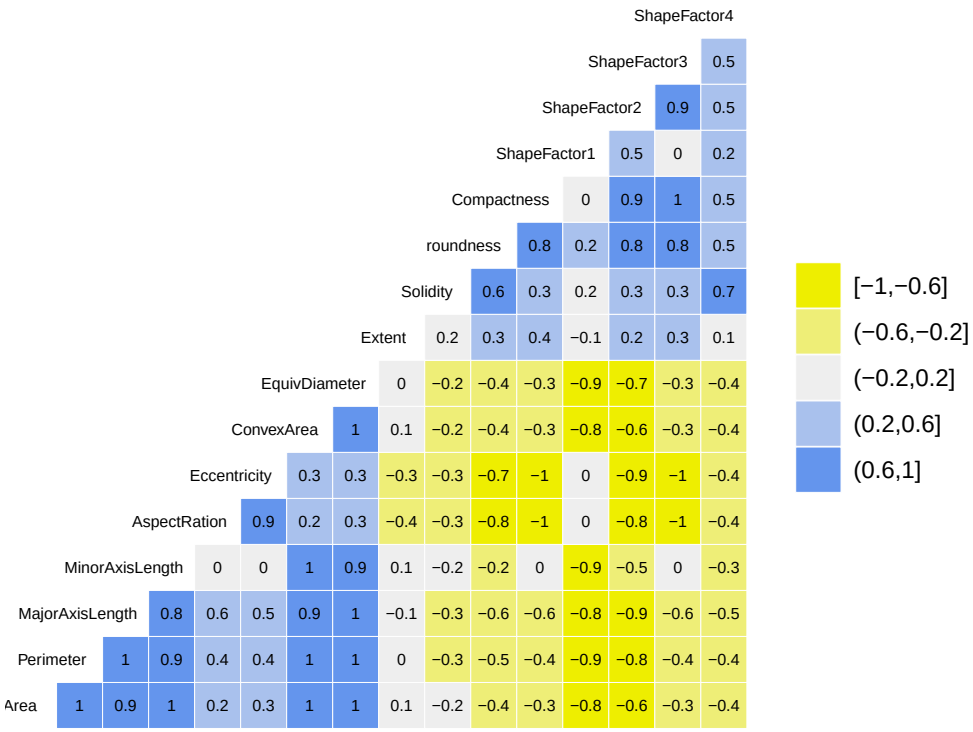
```
mean(db$y == "BARBUNYA")
```

```
## [1] 0.09712732
```

One imagines that the classes that are least represented would be the easiest to sort. We hope!

The next step I will take is to make a correlation matrix of the features in a step-wise plot display. I assume that there will be strong correlation between some of the features, especially the geometric measurements (which should be collinear).

```
#Correlation matrix using Pearson method, default method is Pearson
db$x %>% ggcorr(high = "cornflowerblue",
                low = "yellow2",
                label = TRUE,
                hjust = .9,
                size = 2,
                label_size = 2,
                nbreaks = 5) +
  labs(title = "Correlation Matrix",
       subtitle = "Pearson Method Using Pairwise Obervations")
```



As we would expect, we can see there are some very high correlations in features, and we even see multi-collinearity. This is a term that means many of the features we would use for predictive purposes are collinear.

The main reason I manipulated the features into a matrix is so that I could pre-process the data before composing classification models. First I will check to see which features have the highest mean and the lowest standard deviation, and then I will center and scale the x features.

```
# Which feature has the highest mean
which.max(colMeans(db$x))
```

```
## ConvexArea
##          7
```

```
# Which feature has the lowest standard deviation
which.min(colSds(db$x))
```

```
## [1] 14
```

```
# Centers and scales the data
x_centered <- sweep(db$x, 2, colMeans(db$x))
x_scaled <- sweep(x_centered, 2, colSds(db$x), FUN = "/")

sd(x_scaled[,1])
```

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

```
median(x_scaled[,1])
```

```
## [1] -0.2863271
```

Now that I have a scaled version of the features, it is time for some distance calculations. The following

calculates the distance between all samples using the scaled matrix. Then it calculates the distance between the first sample, which is classified as Seker, and other Seker samples. Finally, it calculates the distance from the first Seker sample to Bombay samples.

```
d_samples <- dist(x_scaled)

dist_SEtoSE <- as.matrix(d_samples)[1, db$y == "SEKER"]
mean(dist_SEtoSE[2:length(dist_SEtoSE)])
```

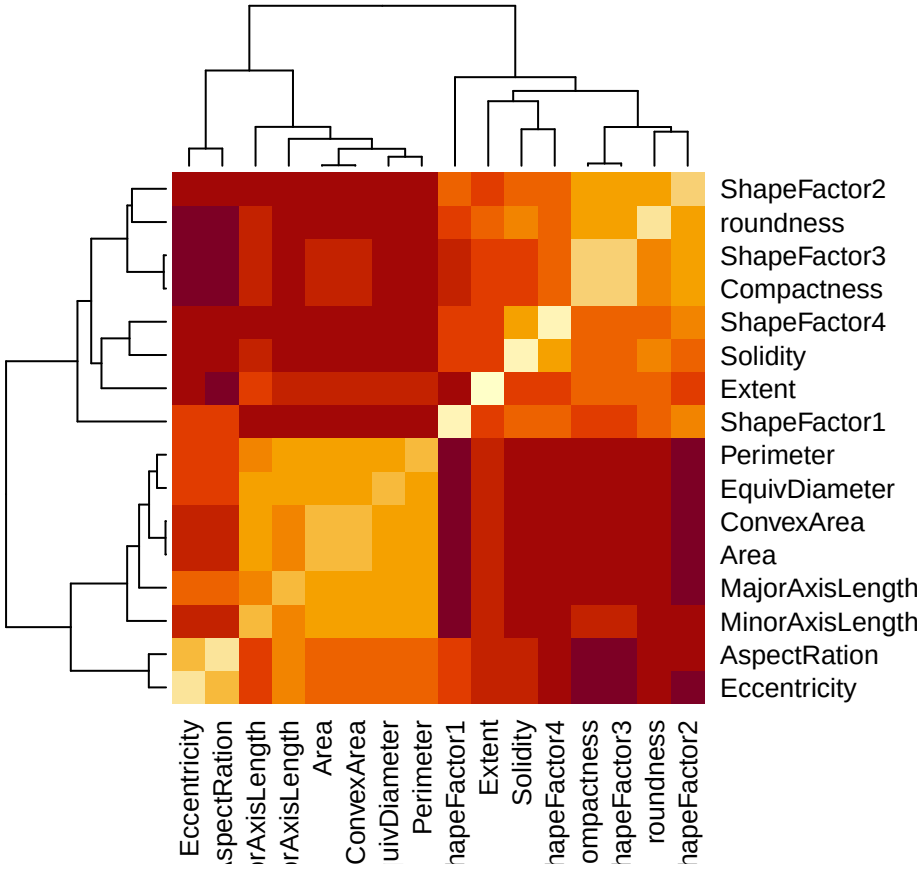
```
## [1] 2.404697
```

```
dist_SEtoBO <- as.matrix(d_samples)[1, db$y == "BOMBAY"]
mean(dist_SEtoBO)
```

```
## [1] 13.57055
```

Let's make a heatmap of the relationship between features using the scaled matrix.

```
d_features <- dist(t(x_scaled))
heatmap(as.matrix(d_features))
```



So, here we have another useful display of the relationship between features.

The next step will perform hierarchical clustering on the 16 features. I will cut the tree into 5 groups.

```
h <- hclust(d_features)
groups <- cutree(h, k = 5)
split(names(groups), groups)
```

```
## $`1`
```

```
## [1] "Area"            "Perimeter"       "MajorAxisLength" "MinorAxisLength"
## [5] "ConvexArea"      "EquivDiameter"
##
## $`2`
## [1] "AspectRation" "Eccentricity"
##
## $`3`
## [1] "Extent"
##
## $`4`
## [1] "Solidity"     "roundness"    "Compactness"  "ShapeFactor2" "ShapeFactor3"
## [6] "ShapeFactor4"
##
## $`5`
## [1] "ShapeFactor1"
```

I could use this clustering as a base-point for a dimensional reduction of features. However, I would like to continue with more analysis.

Now, because we have so many collinear features, I need to perform principal component analysis. The results of which will serve as a form of feature reduction for the machine learning I will employ later on in the project.

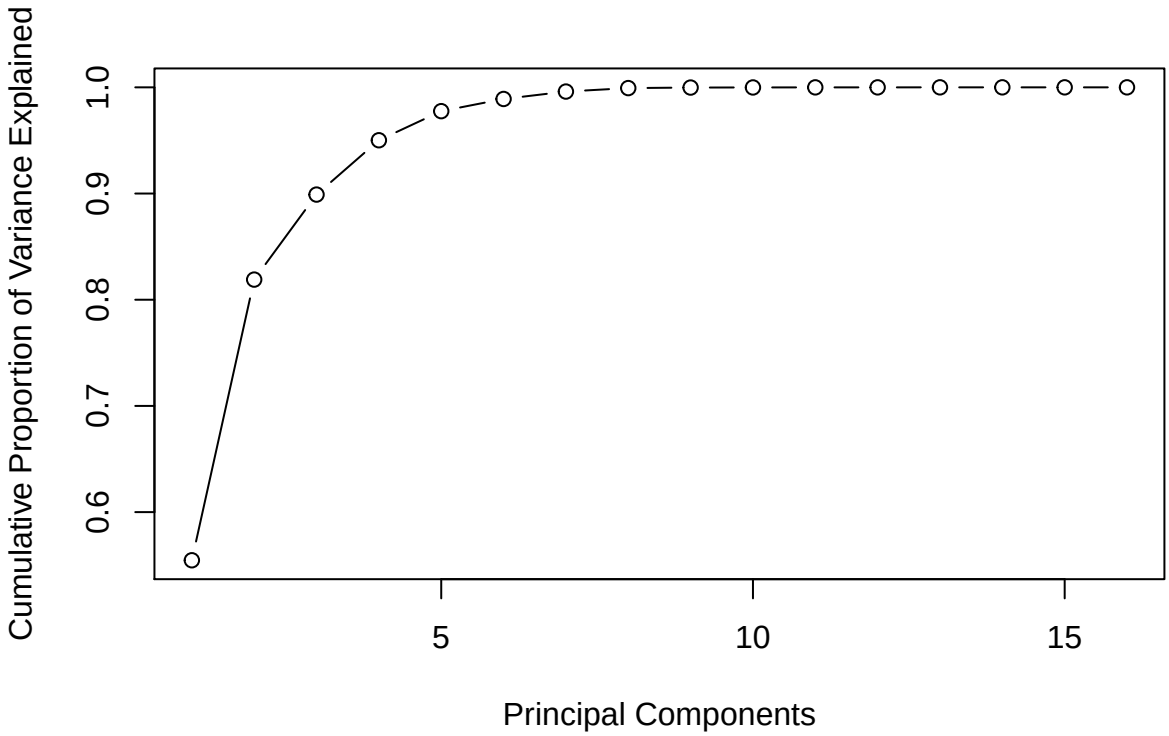
```
pca <- prcomp(x_scaled)
pca_sum <- summary(pca)
pca_sum$importance
```

```
##                             PC1      PC2      PC3       PC4       PC5       PC6
## Standard deviation     2.979032 2.056442 1.131835 0.9045733 0.6620324 0.4289076
## Proportion of Variance 0.554660 0.264310 0.080070 0.0511400 0.0273900 0.0115000
## Cumulative Proportion  0.554660 0.818970 0.899040 0.9501800 0.9775700 0.9890700
##                             PC7      PC8        PC9       PC10       PC11
## Standard deviation     0.334102 0.228064 0.09088598 0.03812991 0.03246827
## Proportion of Variance 0.006980 0.003250 0.00052000 0.00009000 0.00007000
## Cumulative Proportion  0.996050 0.999300 0.99981000 0.99991000 0.99997000
##                            PC12       PC13        PC14        PC15
## Standard deviation     0.01714593 0.01219814 0.003163901 0.001464962
## Proportion of Variance 0.00002000 0.00001000 0.000000000 0.000000000
## Cumulative Proportion  0.99999000 1.00000000 1.000000000 1.000000000
##                              PC16
## Standard deviation     0.001335961
## Proportion of Variance 0.000000000
## Cumulative Proportion  1.000000000
```

95% of the variance is explained by the first 4 principal components, and more than half of proportion of variance is explained in the first principal component alone.

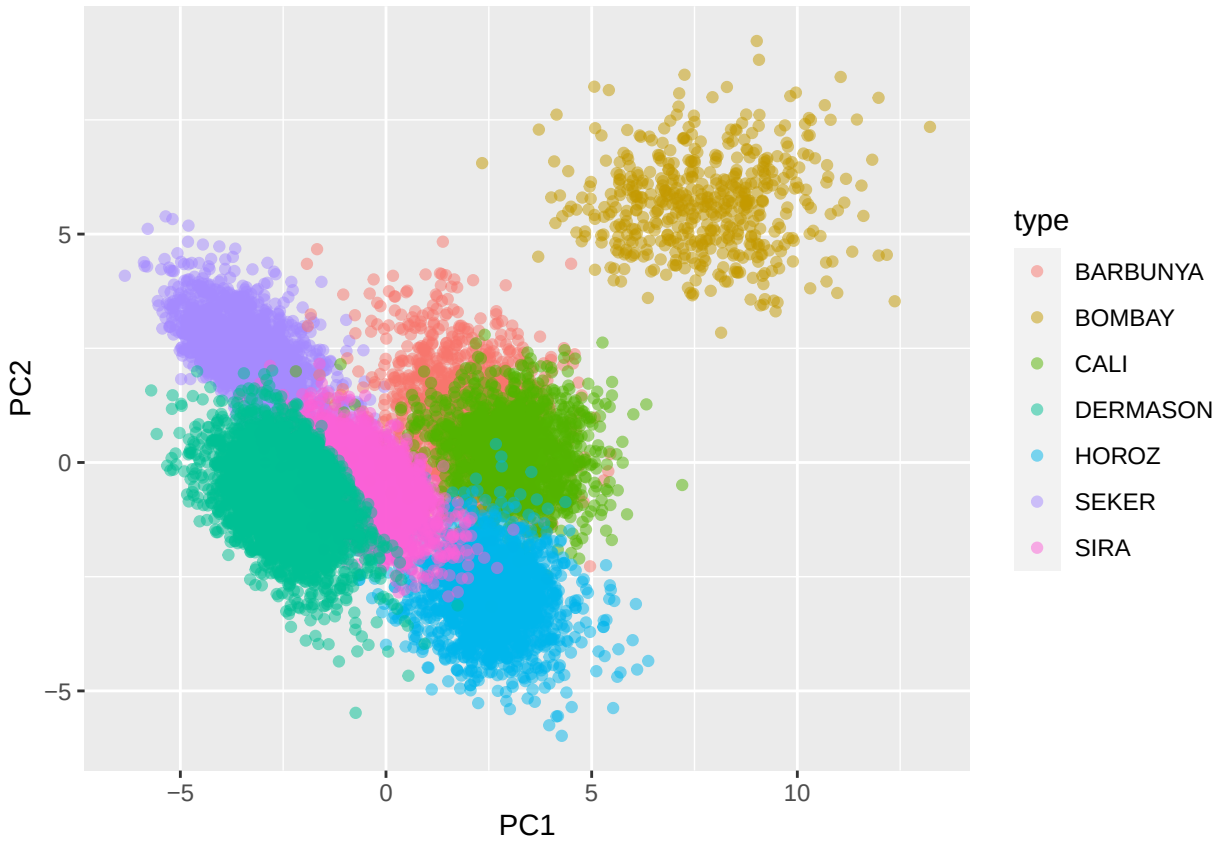
Time to make a plot and visualize what this looks like.

```
var_explained <- cumsum(pca$sdev^2/sum(pca$sdev^2))

plot(var_explained, xlab = "Principal Components",
     ylab = "Cumulative Proportion of Variance Explained", type = "b")
```



Now I want to visualize the delineation between the classes (type of bean) by plotting the first two principal components with color representing the bean types.

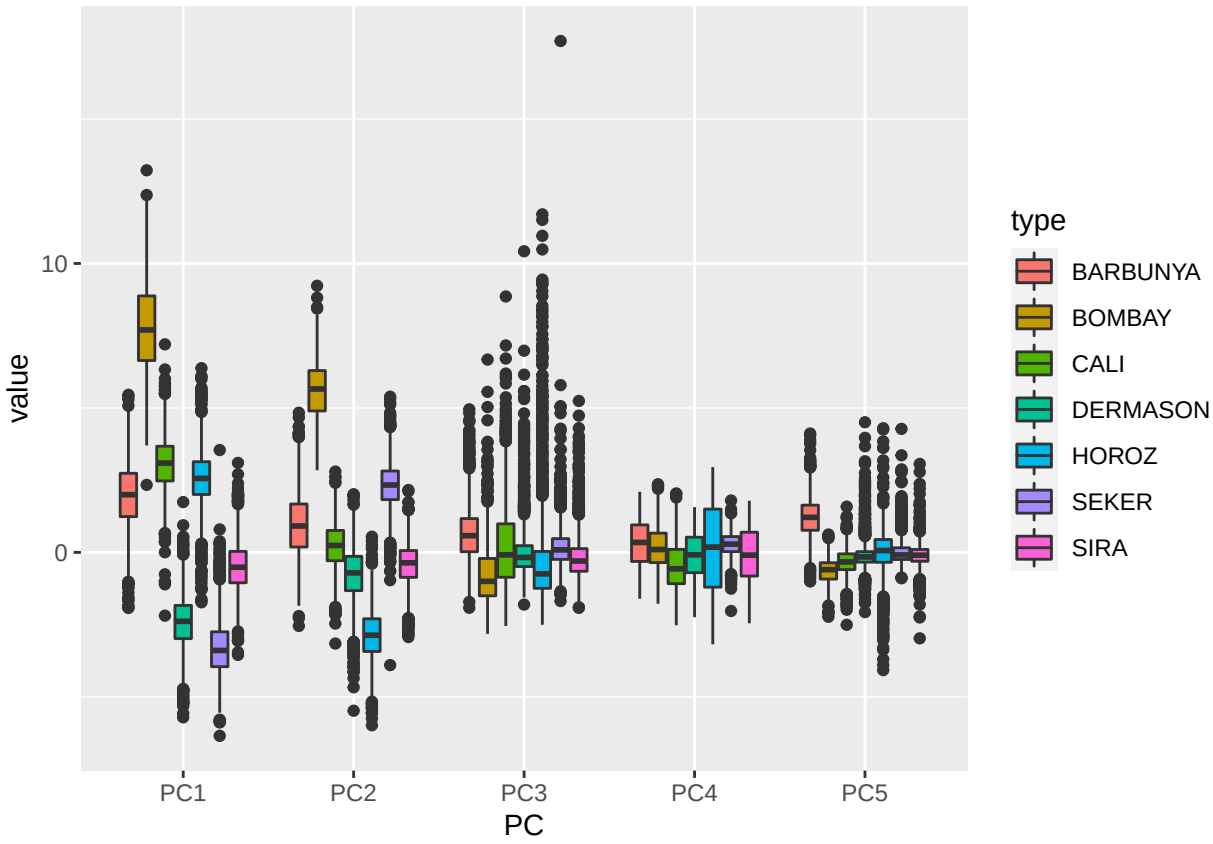
```
data.frame(pca$x[,1:2], type = db$y) %>%
  ggplot(aes(PC1, PC2, value, color = type)) +
  geom_point(alpha = .5)
```



So, this plot tells us that Bombay beans tend to have very high values of PC1 and PC2. It also tells us, among other things, that Sira beans have a similar spread of values for PC1 and PC2.

Now I will make a boxplot of the first 5 PCs grouped by bean type

```
data.frame(type = db$y, pca$x[,1:5]) %>%
  gather(key = "PC", value = "value", -type) %>%
  ggplot(aes(PC, value, fill = type)) +
  geom_boxplot()
```



The following creates a new object using the results from the principal component analysis so that we can proceed to the machine learning phase. This is a form of feature reduction that allows us to be certain each feature is providing helpful information for the predictive process. I have chosen to include the first 7 principal components because they explain 99% of the variance between classes.

```
db <- with(db, list(x =as.matrix(pca$x[,1:7]), y = as.factor(db$y)))
```

Split the data

The following sets the seed to 1, then creates a data partition splitting y and the reduced pca matrix into 20% test and 80% train sets.

I will use cross validation in the training set to report accuracy until I select the best model to run a true validation with the test set data.

```
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(db$y, times = 1, p = 0.2, list = FALSE)
test_x <- db$x[test_index,]
test_y <- db$y[test_index]
train_x <- db$x[-test_index,]
train_y <- db$y[-test_index]
```

Models

I will propose a number of models, and test for accuracy. I will append each model's accuracy to a table in order to keep a running list for comparisons.

LDA model

```
set.seed(12, sample.kind = "Rounding")
train_lda <- train(train_x, train_y, method = "lda",
                   trControl = trainControl(method = "cv", number = 5 ))

# reported accuracy from the cross validation in the train set only
lda_acc <- train_lda$results[,2]

Accuracy_Results <- tibble(Method = "LDA", Accuracy = lda_acc)
Accuracy_Results %>% knitr::kable()
```

Method	Accuracy
LDA	0.9000479

QDA model

```
set.seed(312, sample.kind = "Rounding")
train_qda <- train(train_x, train_y, method = "qda",
                   trControl = trainControl(method = "cv", number = 5 ))

# reported Accuracy from the cross validation in the train set only
qda_acc <- train_qda$results[,2]

Accuracy_Results <- bind_rows(Accuracy_Results,
                              tibble(Method = "QDA", Accuracy = qda_acc))
Accuracy_Results %>% knitr::kable()
```

Method	Accuracy
LDA	0.9000479
QDA	0.9173178

Caret's treebag

Bagging (Bootstrap Aggregating) algorithms are used to improve model accuracy in regression and classification problems. The basic idea is that they build multiple models from separated subsets of training data, and then construct an aggregated and more accurate final model.

I'll use Caret's treebag version of the bagging method for classification.

```
set.seed(9874, sample.kind="Rounding")
trCtrl <- trainControl(method = "cv", number = 5)
cr.fit <- train(train_x, train_y,
                method  = "treebag",
                trControl = trCtrl,
                metric = "Accuracy")

# reported Accuracy from the cross validation in the train set only
crtb_acc <- cr.fit$results[,2]

Accuracy_Results <- bind_rows(Accuracy_Results,
                              tibble(Method = "TREEBAG", Accuracy = crtb_acc))
```

```
Accuracy_Results %>% knitr::kable()
```

Method	Accuracy
LDA	0.9000479
QDA	0.9173178
TREEBAG	0.9190648

Now let's turn to support vector machines.

Support Vector Machines

I found an informative booklet which gives an overarching look into the mathematics at work in SVM's online. It is called "An Idiot's guide to Support vector machines (SVMs)" by R. Berwick. You can read it by following the link below. http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf

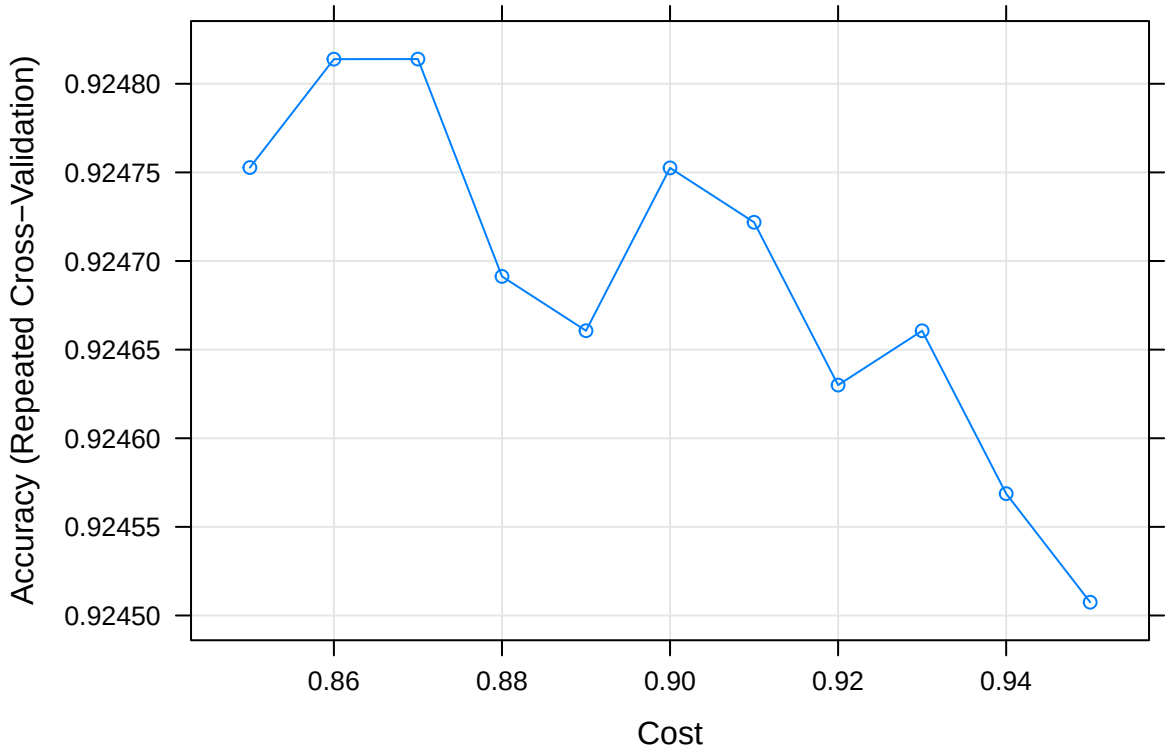
The two types I will use are linear grid and radial kernel.

Linear Grid Support Vector Model

```
#############################################################################
## IMPORTANT NOTE - I used cloud computing on AWS
# this TAKES 3 minutes for my setup which had a 32 thread CPU, 32gb ram and a
# gpu with 8gb of dedicated memory. It didn't come close to taxing anything.
# Therefore this should be safe to run locally.
#
# It may take you a longer run time.  This is because my code
# uses repeated cross validation to choose the Cost parameter that
# maximize the model accuracy.
set.seed(56543, sample.kind="Rounding")
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
grid <- expand.grid(C = seq(0.85, .95, 0.01))
svm_Linear_Grid <- train(train_x, train_y,
                         method = "svmLinear",
                         trControl=trctrl,
                         tuneGrid = grid,
                         tuneLength = 10)

# Print the best tuning parameter C that
# maximizes model accuracy

svm_Linear_Grid$bestTune
```

```
##      C
## 3 0.87
```

```r
plot(svm_Linear_Grid)
```




```r
# reported Accuracy from the cross validation in the train set only
svm_LG_Acc <- svm_Linear_Grid$results[3,2]

Accuracy_Results <- bind_rows(Accuracy_Results,
                              tibble(Method = "LinearSVM",
                                     Accuracy = svm_LG_Acc))
Accuracy_Results %>% knitr::kable()
```

Method	Accuracy
LDA	0.9000479
QDA	0.9173178
TREEBAG	0.9190648
LinearSVM	0.9248139

Radial Support Vector Model

The cross validation of the training data in this model takes some time.

```r
#############################################################################
## IMPORTANT NOTE - I used cloud computing on AWS
# This TAKES 8 minutes for my setup which had a 32 thread CPU.
# This is safe to run locally, but will take quite some time if you do not
# have a modern multi-core cpu.

set.seed(1985, sample.kind = "Rounding")
radial_svm <- train(train_x, train_y,
                    method = "svmRadial",
```

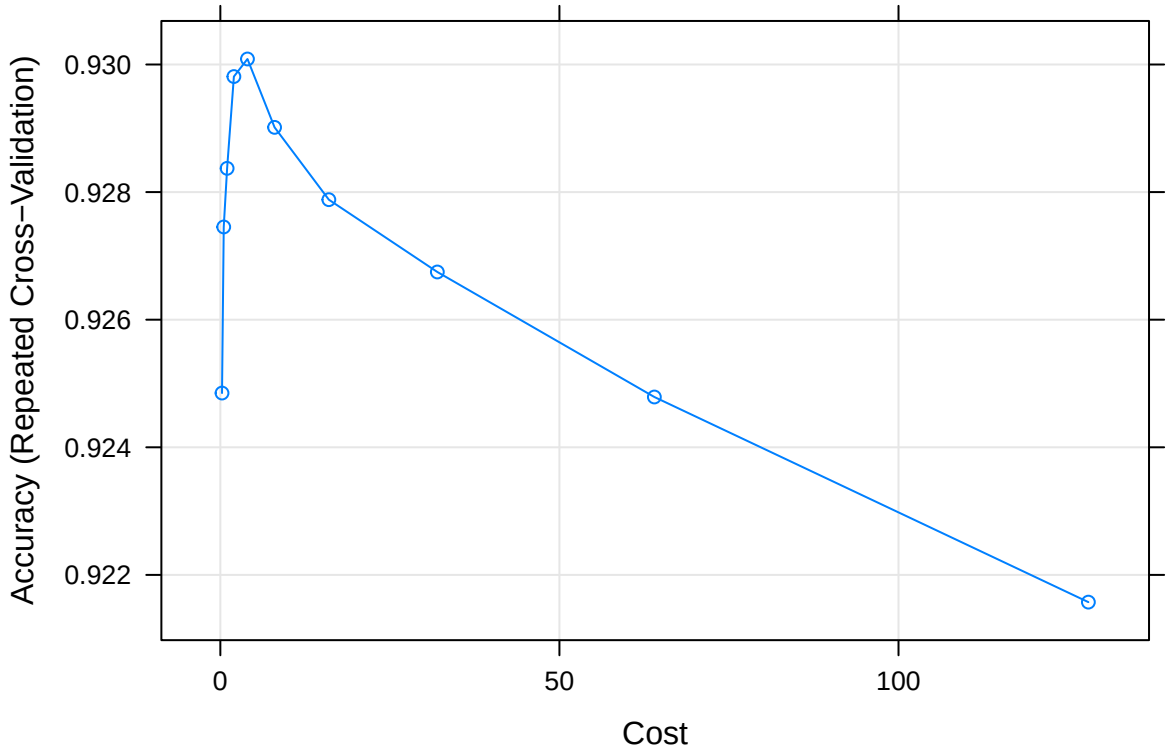
```
                    trControl = trainControl("repeatedcv",
                                             number = 10, repeats = 3),
                    tuneLength = 10)
```

Time to print the best values for sigma and cost. Then we can plot the model accuracy according to the Cost parameter.

```
radial_svm$bestTune
```

```
##       sigma C
## 5 0.1775565 4
```

```
plot(radial_svm)
```



Now we can report the accuracy to our results table.

```
# reported Accuracy from the cross validation in the train set only
radial_svm_acc <- radial_svm$results[5,3]

Accuracy_Results <- bind_rows(Accuracy_Results,
                              tibble(Method = "RadialSVM",
                                     Accuracy = radial_svm_acc))
Accuracy_Results %>% knitr::kable()
```

Method	Accuracy
LDA	0.9000479
QDA	0.9173178
TREEBAG	0.9190648
LinearSVM	0.9248139
RadialSVM	0.9300866

We have reached the highest accuracy yet of around 93% from the cross validation in the train set.

Random Forest Model

Next up I will train a random forest model and see how it compares. Because this takes an intolerable amount of time to train without parallel processing, I do not recommend trying this on your personal machine. I used an AWS server with 32 threads and 64 gb of Ram. I commented in the code below how much Ram was consumed during this process. However, it makes the run time exponentially shorter. DO NOT RUN UNLESS YOU ARE CERTAIN YOUR MACHINE IS CAPABLE!

```
######################## DO NOT RUN ################################################
#    unless you are certain your machine is capable          ###################
#   With 32 threads in CPU it used 34 gb of Ram                    ############
###############################################################################
# The Parallel Computing is why it is demanding in RAM                        #
# But it takes too long to run without it                                     #
###############################################################################
numbCores <- detectCores()-4 # Protects against a crash & makes my core count 28
cl <- makeCluster(numbCores)
registerDoParallel(cl)
#34Gb of Ram used, but it reduced my run-time to 3 minutes!
set.seed(9, sample.kind="Rounding")
tuning <- data.frame(mtry = c(1,2,3,4,5))
train_rf <- train(train_x, train_y,
                  method = "rf",
                  tuneGrid = tuning,
                  importance = TRUE)
```

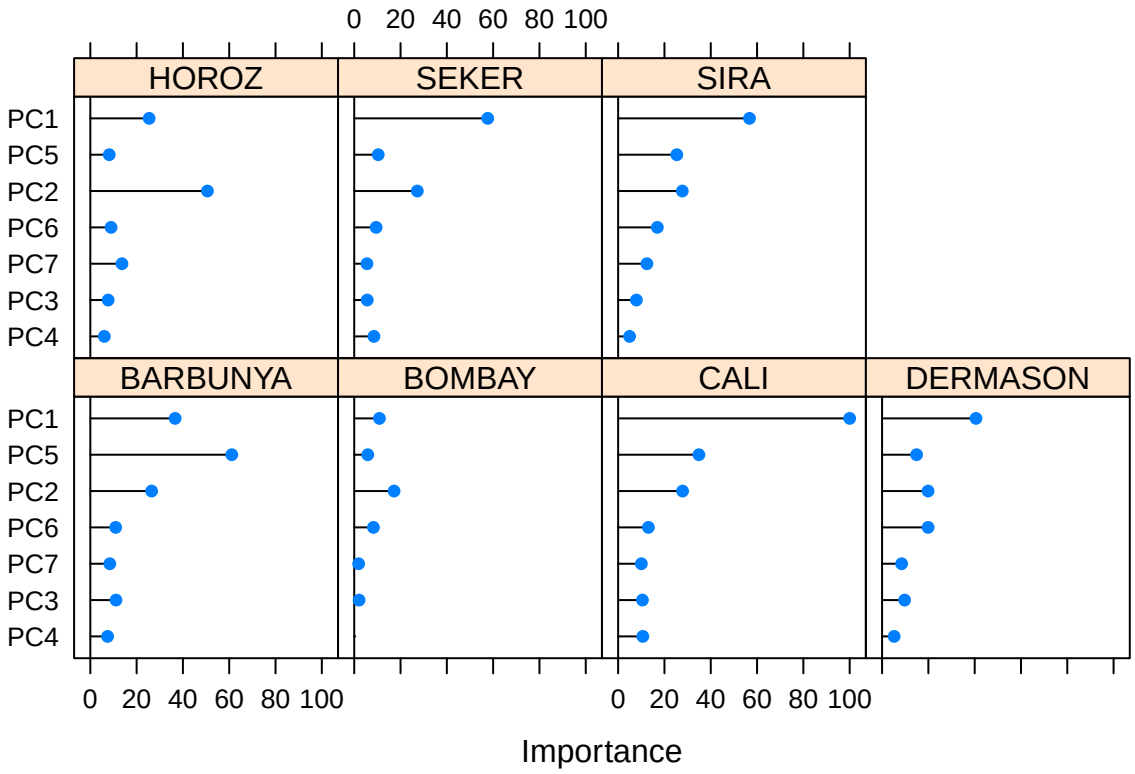
Show the best tune.

```
train_rf$bestTune
```

```
##   mtry
## 2    2
```

Next, we take a look at the variable importance.

```
plot(varImp(train_rf))
```



The code below appends the accuracy of the best tune from the cross validation in the train set to our running table.

```
# Report the accuracy of the best tune from cross validation in the train set
rf_acc <- train_rf$results[2,2]
Accuracy_Results <- bind_rows(Accuracy_Results,
                              tibble(Method = "RandomForest",
                                     Accuracy = rf_acc))
Accuracy_Results %>% knitr::kable()
```

Method	Accuracy
LDA	0.9000479
QDA	0.9173178
TREEBAG	0.9190648
LinearSVM	0.9248139
RadialSVM	0.9300866
RandomForest	0.9241889

KNN Model

```
set.seed(234, sample.kind="Rounding")
tuning <- data.frame(k = seq(20, 30, 1))
train_knn <- train(train_x, train_y,
                   method = "knn",
                   tuneGrid = tuning)
```

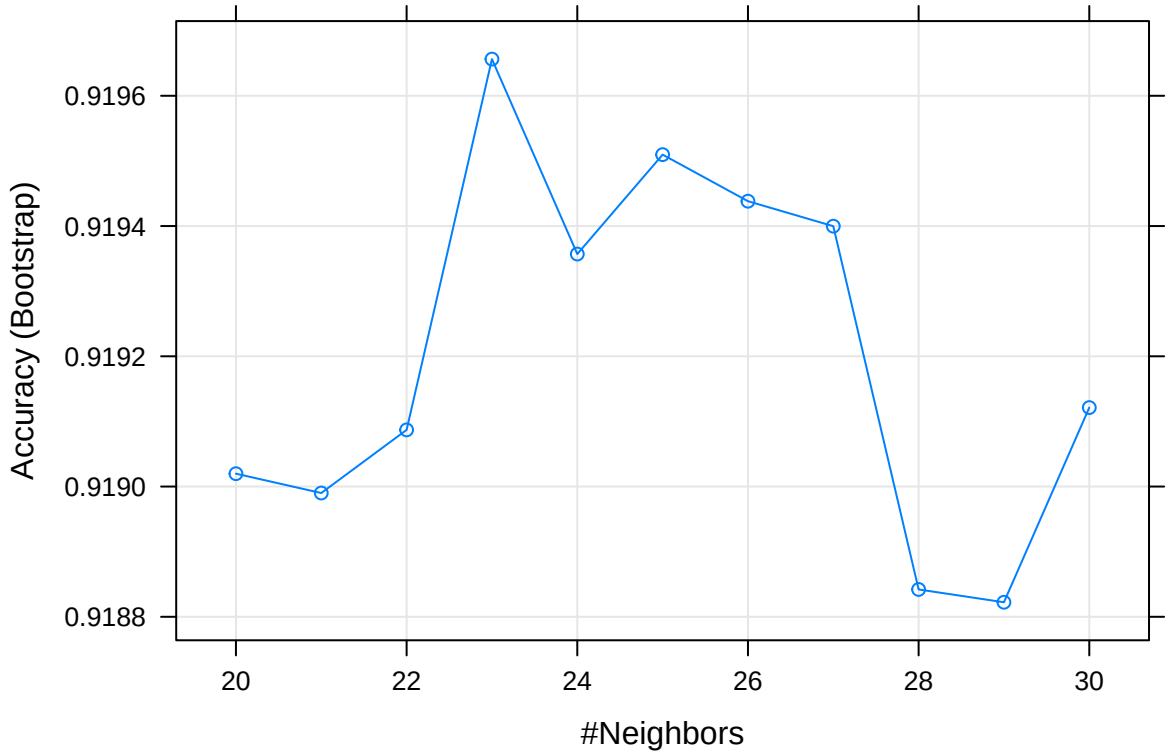
Pull the best tune.

```
train_knn$bestTune
```

```
##    k
## 4 23
```

Plot the model accuracy of each k neighbors.

```
plot(train_knn)
```



The following reports the accuracy from the cross validation in the train set.

```
# report the accuracy from the cross validation in the train set
knn_acc <- train_knn$results[4,2]


Accuracy_Results <- bind_rows(Accuracy_Results,
                              tibble(Method = "knn", Accuracy = knn_acc))
Accuracy_Results %>% knitr::kable()
```

Method	Accuracy
LDA	0.9000479
QDA	0.9173178
TREEBAG	0.9190648
LinearSVM	0.9248139
RadialSVM	0.9300866
RandomForest	0.9241889
knn	0.9196563

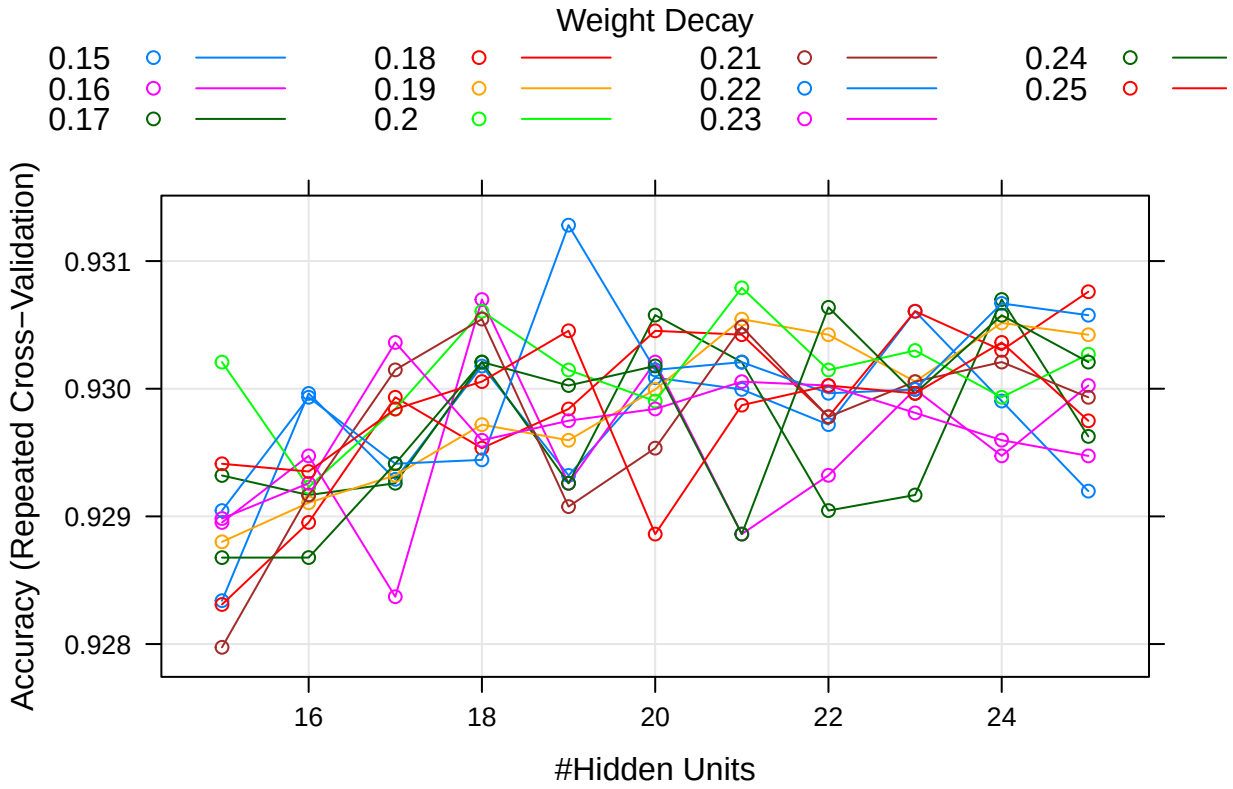
Neural Network

The following runs a neural network that uses cross validation in the train set to optimize the hidden layers and decay in order to maximize the model accuracy.

```
#####################  Warning  ###################################################
# The parallel cluster is still running for this model
# It is not a ram heavy model and can be run with a regular modern cpu
# Just be prepared to wait and wait and ... wait if you do not implement the
# doParallel library as shown above

# One more reminder - I used AWS with a 32 CORE CPU - and the parallel
# computations still take a few minutes
################################################################################

set.seed(2976, sample.kind = "Rounding")
# I will use cross validation in the train set in order to find the optimal
# hidden layers and decay.
tc <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
cv_nn <- train(train_x, train_y, method='nnet', linout=TRUE, trace = FALSE,
                  trControl = tc,

                  #Grid of tuning parameters to try:
                  tuneGrid=expand.grid(.size= seq(15,25,1),.decay=seq(0.15,0.25,0.01)))

# Examine the results with a plot
cv_nn$bestTune
```

```
##    size decay
## 52   19  0.22
```

```
plot(cv_nn)
```




```
# report the accuracy from the cross validation and append it to the running
# table
nn_acc <- cv_nn$results[52,3]

# Appends results to the table
Accuracy_Results <- bind_rows(Accuracy_Results,
                              tibble(Method = "nnet", Accuracy = nn_acc))
Accuracy_Results %>% knitr::kable()
```

Method	Accuracy
LDA	0.9000479
QDA	0.9173178
TREEBAG	0.9190648
LinearSVM	0.9248139
RadialSVM	0.9300866
RandomForest	0.9241889
knn	0.9196563
nnet	0.9312814

So we have clear leader with the neural net using cross validation in the train set. It is now time for the final validation.

Final Validation - NNet

```
# use the cross validated neural net model to run the predictions on the test
# set
preds_nn <- predict(cv_nn, test_x)
```

```
# Calculates Accuracy
final_Val <- tibble(Method = "Final Val - NNet", Accuracy = mean(preds_nn==test_y))
final_Val
```

```
## # A tibble: 1 x 2
##   Method           Accuracy
##   <chr>               <dbl>
## 1 Final Val - NNet    0.939
```

We have beaten the research paper and have a final validation with accuracy at around 93.87%.

Before we move on to the summary section it is important to stop the cluster for parallel computing.

```
###stop cluster
stopCluster(cl)
```

SUMMARY

In summary, I have met my goal of an accuracy above 93%. The most accurate model was the Neural Network Model which used cross validation in the train set to select the optimal size of the hidden layers and decay for maximizing accuracy.

Let's have a look at a confusion matrix for the final prediction on the test set.

```
confusionMatrix(preds_nn,test_y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
##   BARBUNYA      251      0    7        0     1     1    3
##   BOMBAY          0    105    0        0     0     0    0
##   CALI            9      0  313        0     6     0    0
##   DERMASON        0      0    0      664     6     8   42
##   HOROZ           0      0    4        4   364     0    5
##   SEKER           2      0    0        9     0   389    5
##   SIRA            3      0    2       33     9     8  473
##
## Overall Statistics
##
##                Accuracy : 0.9387
##                  95% CI : (0.9291, 0.9474)
##     No Information Rate : 0.2605
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9259
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BARBUNYA Class: BOMBAY Class: CALI Class: DERMASON
## Sensitivity                  0.94717       1.00000      0.9601          0.9352
## Specificity                  0.99512       1.00000      0.9938          0.9722
## Pos Pred Value               0.95437       1.00000      0.9543          0.9222
```

```
## Neg Pred Value               0.99432       1.00000      0.9946          0.9771
## Prevalence                   0.09721       0.03852      0.1196          0.2605
## Detection Rate               0.09208       0.03852      0.1148          0.2436
## Detection Prevalence         0.09648       0.03852      0.1203          0.2641
## Balanced Accuracy            0.97115       1.00000      0.9769          0.9537
##                       Class: HOROZ Class: SEKER Class: SIRA
## Sensitivity                 0.9430       0.9581      0.8958
## Specificity                 0.9944       0.9931      0.9750
## Pos Pred Value              0.9655       0.9605      0.8958
## Neg Pred Value              0.9906       0.9927      0.9750
## Prevalence                  0.1416       0.1489      0.1937
## Detection Rate              0.1335       0.1427      0.1735
## Detection Prevalence        0.1383       0.1486      0.1937
## Balanced Accuracy           0.9687       0.9756      0.9354
```

This gives us some helpful information. We have an overall accuracy rate of 93.87%, with balanced accuracy rates above 95% across all classes with the sole exception of the the bean class Sira (93.54%).

That being said there are some limitations to this type of classification. As stated earlier, the measurements of the features come from a computer vision system which makes the delineation between classes cumbersome due to the collinear features. We saw how scaling the features and running principal component analysis aided in the process of feature dimension reduction. The models were trained to a good level of accuracy as a result.

If the market demands a more accurate classification system for the dry grains, perhaps another feature, which might not be so highly correlated, could be measured to aid the process? For instance, if I had access to the images, I would want to examine color saturation levels between the dry beans. Perhaps it could have been of some use for the process by providing a feature measurement that is not so highly correlated?

I hope you have enjoyed reading this project.