Stat602 Final Group Project

Rylie Fleckenstein, Seema Bhandari, Yamuna Dhungana, Roshni Sharma, Lilibeth Lumbreras

5/4/2021

Typically, cultivation of dry bean seeds in Turkey yields mixed varieties during harvest. This poses a problem as market value of dry beans are dependent on whether there is separability between the different varieties of beans (Koklu and Ozkan, 2020). Upon evaluation at market, the net worth of a sample of beans decreases if there is no clear identification between the beans. In this study, our task is to develop an automated method that can ameliorate the issue of decreased market values of dry beans by predicting the value of a harvest from a 'population cultivation' from a single farm that has been presented at market. Given three different sets of single pound of white beans, our method will predict the classifications of the bean varieties and each set's estimated net worth. Additional measures of confidence and uncertainty for our estimates will be provided to support those predictions.

From our discussions, our team developed a model building process to support a solution for the dry bean problem. Prior to initiation of the model building process, each group member chose a classification technique to test. The formal approach first requires exploratory analysis of the data. This analysis of the data assists in identifying any initial patterns or correlations in the labeled dataset. The second step involves pre-processing the data prior to testing. For some techniques, they require pre-processing the data by dummy encoding target variables or scaling the data. Next, the process involves splitting the labeled data into training and testing groups. Our team decided that we would use an 80/20 training and testing split for our analysis. In our fourth step, we tested our models using our chosen classification techniques. The models are built using the training group and then tested using the testing split of the labeled data. In the next step, we used our own error determination algorithm and formula to calculate metrics of price value accuracy. In our sixth step, we then applied the algorithm and formula to the three sample sets. In our seventh step, we used metrics like overall training accuracy, F1-score, and the price value accuracies applied to the samples to determine model selection and our final recommendation.

The labeled dataset consists of 3000 observations and eight variables: Class (our dependent variable), Area, Perimeter, MajorAxisLength, MinorAxisLength, Eccentricity, ConvexArea, and Extent. The first thing we wanted to do was take a look at the predictors and see if there were any strong correlations between them. As we can see in *Table 1*, the correlation matrix, several of the predictors are strongly correlated which we will be able to see through some visualizations. In the plots shown in *Plots 1* and *Plots 2* we take a look at the different predictors just to get an idea if the different classes of the response variable are separable and if using a machine learning approach is the right choice. From the data visualizations it appears we will be able to build a reliable classifier for this problem with the existing predictor variables.From Koklu and Ozkan 2020, we noted that the original study already identified the 12 most effective dimensional features of beans for classification. Our current task consists of using a subset of seven of those identified effective features, so we felt no additional variable selection was necessary. Some data pre-processing is necessary to prepare the data for use in our classifiers. In order to build a neural network with the 'neuralnet' library we must dummy encode the target variable which is shown in *Figure 1.* The last step of data pre-processing involves scaling our data. In order for a few of our algorithms to function properly we must make sure all of our data is normalized. We will create a separate normalized data set to be used in the algorithms that require it, and keep the original data for the algorithms that don't. Finally, for our training and test data sets used for building our classifiers, our team decided to use an 80/20 split.

We constructed our neural network, consisting of 3 hidden layers. The first hidden layer has 15 nodes, the second hidden layer has 10 nodes, and the third hidden layer has 3 nodes. The input layer has a node for each variable so there are 7 and the output layer has a node for each class in the target variable so there are 6. We have set the parameter ‘linear.output’ to FALSE in order to let the network know that the activation function should be applied to the output neurons. This is the case because we are dealing with a multi-class classification problem and we want the network to generate predicted probabilities for each class. The network achieves this by running the output through the activation function that we have set which in this case is the ‘logistic’ function. We then take the neuron with the highest probability and predict that class to be the class for that bean. The final model is trained using our entire “labeled” data set and is to be used for making predictions on our samples (www.rdocumentation.org/packages/neuralnet).

Our second model, Linear Discriminant Analysis (LDA), attempts to maximize the separation between classes. Using the maximized linear combination of predictor variables, the LDA method will predict the classification of the testing data using the lda function from the MASS library. LDA assumes that the predictors are normally distributed, have class specific means, and are of equal variance. In order to verify the accuracy of our estimated test error, we applied Leave-One-Out-Cross-Validation (LOOCV) to the LDA model using all the labeled data by setting CV = TRUE when fitting our model (Saunders, 2018). The mean accuracy was similar to the test accuracy.

The third model we chose to implement was built using the k-nearest neighbors algorithm. The KNN algorithm is a supervised, non-parametric method used for classification and sometimes regression. When used for classification, like the problem we are applying it to, the algorithm returns the predicted class label of each test instance based upon the most prevalent training class of its k closest neighbors. Because this is a distance based algorithm, scaling our data must be part of the preprocessing. In order to build our KNN model we used the knn() function, which is part of the library ‘class’ (Gareth, 2013, p.164). After building our model we fine tuned the hyperparameter ‘k’ (*Figure 2*) and determined the optimal k is 16 since it returns the highest accuracy.

Support vector machine (SVM) is a supervised machine learning algorithm used in classification and regression problems. SVM uses a hyperplane that acts as the decision boundary to classify the data. The dataset ‘labeled’ consists of various attributes like area, perimeter, major axis length, etc. In the same data set, we have a Class variable that is used to classify the type of bean for each instance. With the ‘labeled’ dataset, the model is trained with the function ‘train ()’ using the ‘caret’ library. The train function fits the models over various tuning parameters. The data set is scaled and centered with a tuned length of 10. For this model we are implementing a linear kernel. Using the function trainControl, we conducted a 10 fold CV that was repeated 3 times to choose the optimal train control (Gareth, 2013, p.337, 349, 360).

Random forests are ensemble training methods used for classification and regression problems. They are built by combining a multitude of decision trees and their outputs are generated by taking the average prediction made by all of the individual trees. This algorithm is known for being easy to implement as it effectively deals with missing values and categorical variables. Also, scaling of the data is not required before training this model. Here we built our random forest model using the ‘train’ function from the library ‘caret’. Using the function trainControl, we conducted a 10-fold cross validation to choose the optimal train control. After building our model, we identified the count of randomly selected predictor variables that affecting our random forest method. (*Figure 3*). In the variable importance plot (*Figure 4*), we can see the variables that have the most importance in predicting the class of our beans. The most important variables are Area, Eccentricity and MajorAxisLength. The variables Extent and Perimeter have the lowest levels of importance in our predictions (Gareth, 2013, p.319-321).

The next part of the analysis involves determining how much error (in dollars) there might be in our final predicted price. We know that each of our samples is a 1-pound sample of beans. Knowing this, what we need to do is calculate the weight of our sample predictions given by our models and develop some type of algorithm that gives a higher accuracy score to models whose predictions are close to 1 pound and a lower score to those whose predictions are not. The logical reasoning behind this is that, since we know how many beans are in our sample (number of rows in the data), we know how much the sample is supposed to weigh (1 pound) and we know how much on average one bean for each class weighs, there only exists a certain distribution of beans within the sample that maintains all these constraints (*Formulas*).

We took the predicted bean counts and multiplied each count by its respective average weight per bean.

We then took the sum of these weights to determine the weight of our predictions. We then took the absolute value of the difference between the predicted weight and the actual weight (1 pound for the samples) and divided this by the average weight of a bean to give us the predicted number of beans we are off in classification. Since we cannot confidently say which classes these misclassified beans are from, we deiced to take an average of the bean weights and an average of the bean costs (per bean) and use those values for our calculations. From there we multiplied that misclassification rate by the average price of a bean to determine the predicted price error. To determine the price accuracy, we subtracted the price error from the predicted price and then divided that by the predicted price (*Formulas*).

To gauge how well our training models were performing we used Accuracy, F1-score and we took a look at how close the predicted price was to the actual price (*Table 3*). We also looked at Precision by Bean to get an idea of how a test model's performance may influence bean counts (*Table 4*, *Table 6*, *Table 8*, *Table 10*) with Bombay having the best precision and Dermason and Sira beans having low precisions. This coincides with Koklu and Ozkan's observations. We took into consideration the predicted price accuracies each model had on all of the samples and determined that the best model for this problem was the Random Forest model (*Table 5*, *Table 7*, *Table 9*). Although KNN had the better Accuracy and F1-score, we found that it had performed worse when applied to the samples as it went over price and weight constraints in Samples B and C. The Random Forest classifier, when applied to all three samples, remained within price and weight constraints while still having the second best Accuracy and F1-Scores. Overall the Random Forest classifier had a good balance of model performance metrics and was reliable in its application on the sample sets. After comparing our five models, we determined that the Random Forest classifier was the optimal model for our automated method to provide farms for value estimation when presenting their dry bean at market.

Tables/Figures:

Formulas

Formulas:

Price per Bean (class specific):

$$p_i = w_i * r_i$$

where r_i = rate in dollars/pound per class, w_i = weight in pounds per class

Average price of a bean (not class specific):

$$p_b = \frac{\sum_{i=1}^{n} p_i}{n}$$

where n is the number of classes of bean

Average weight of a bean (not class specific):

$$w_b = \frac{\sum_{i=1}^{n} w_i}{n}$$

where n is the number of classes of bean

Prediction weight:

$$W_{pred} = \sum_{i=1}^{n} C_i w_i$$

where C_i = predicted count of beans per class

Error in weight prediction:

$$W_{diff} = |W_{pred} - W_{act}|$$

where W_{act} = actual weight of the sample (1 pound for our samples)

Error Price:

$$Error_{price} = \frac{W_{diff}}{w_b} * c_b$$

, where w_b = average weight of a bean, and p_b = average price of a bean

Predicted Price:

$$P_{pred} = \sum_{i=1}^{n} p_i * C_i$$

where p_i = price for each class of beans in dollars, and C_i = number of beans, by class, predicted to be in the sample by out classifier.

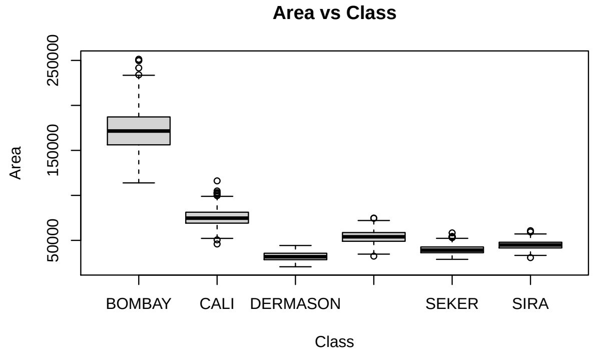
Accuracy:

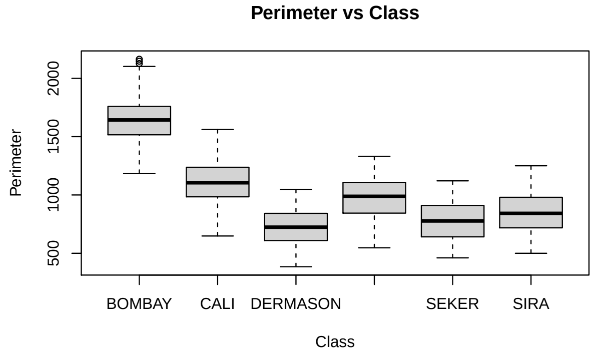
$$Accuracy_{price} = \frac{P_{pred} - Error_{price}}{P_{pred}}$$

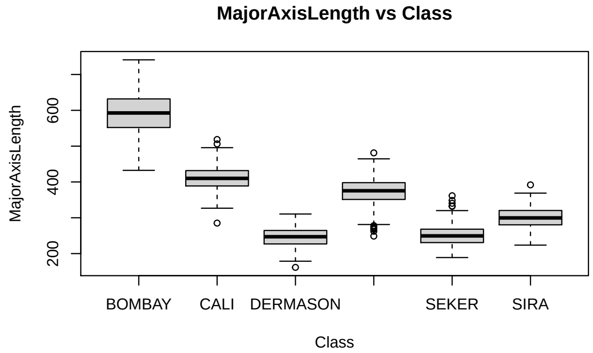
Table 1: Correlation Matrix

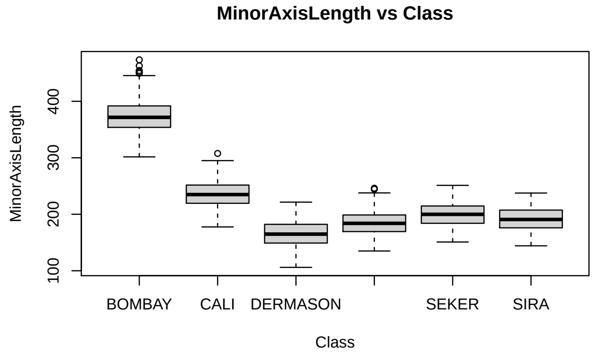
	Area	Perimeter	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent
Area	1.0000000	0.8963858	0.9486155	0.9515599	0.1986577	0.9923247	0.1795214
Perimeter	0.8963858	1.0000000	0.8902704	0.8497052	0.2774474	0.8892845	0.1184731
MajorAxisLength	0.9486155	0.8902704	1.0000000	0.8698992	0.4178660	0.9431600	0.0686936
MinorAxisLength	0.9515599	0.8497052	0.8698992	1.0000000	0.0468264	0.9437090	0.2333211
Eccentricity	0.1986577	0.2774474	0.4178660	0.0468264	1.0000000	0.1998165	-0.2901603
ConvexArea	0.9923247	0.8892845	0.9431600	0.9437090	0.1998165	1.0000000	0.1765168
Extent	0.1795214	0.1184731	0.0686936	0.2333211	-0.2901603	0.1765168	1.0000000

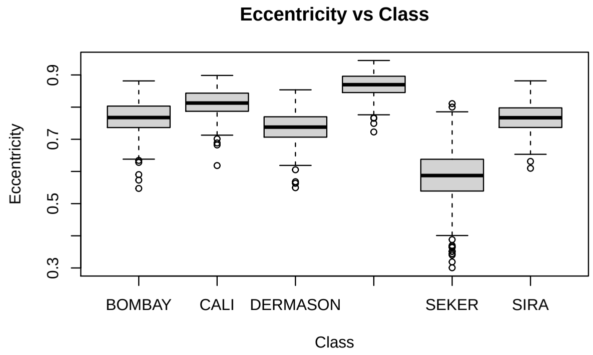
## Plots 1

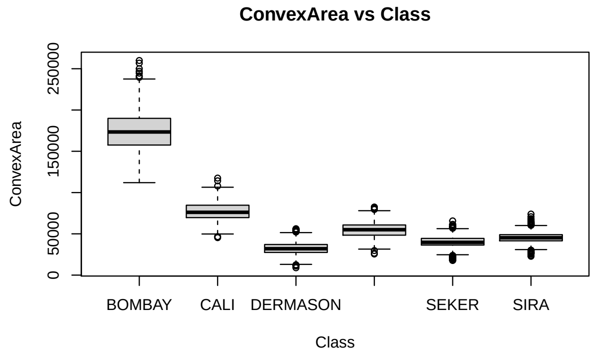

Area vs Class

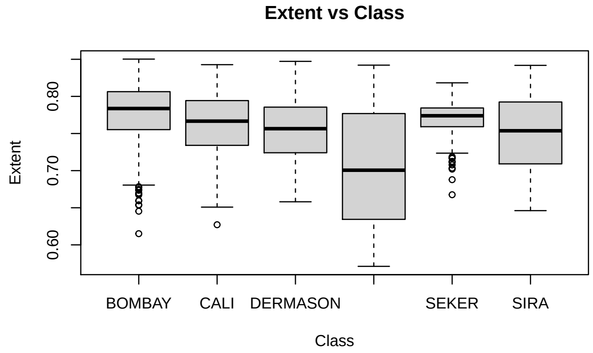

Perimeter vs Class


MajorAxisLength vs Class

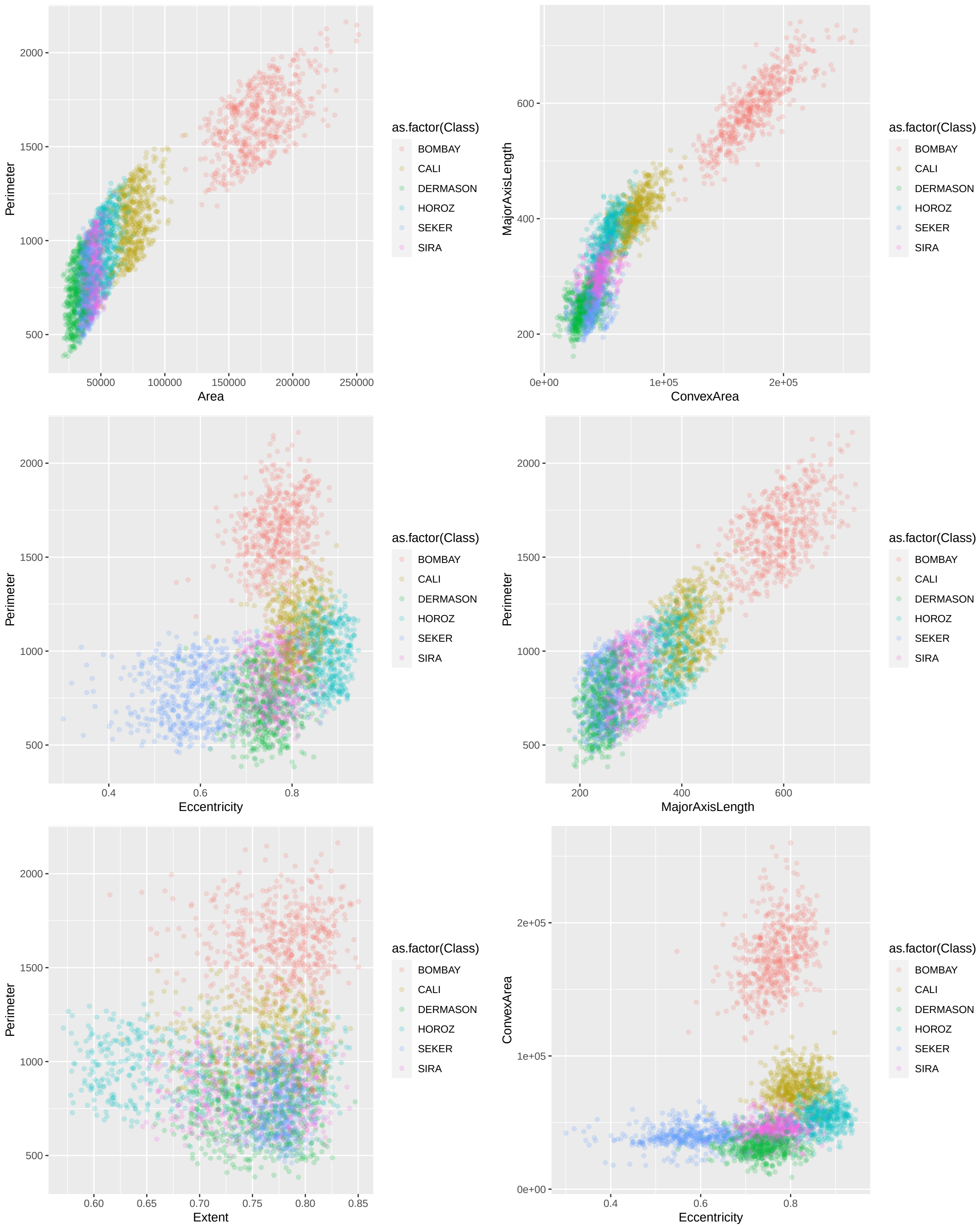

MinorAxisLength vs Class


Eccentricity vs Class


ConvexArea vs Class

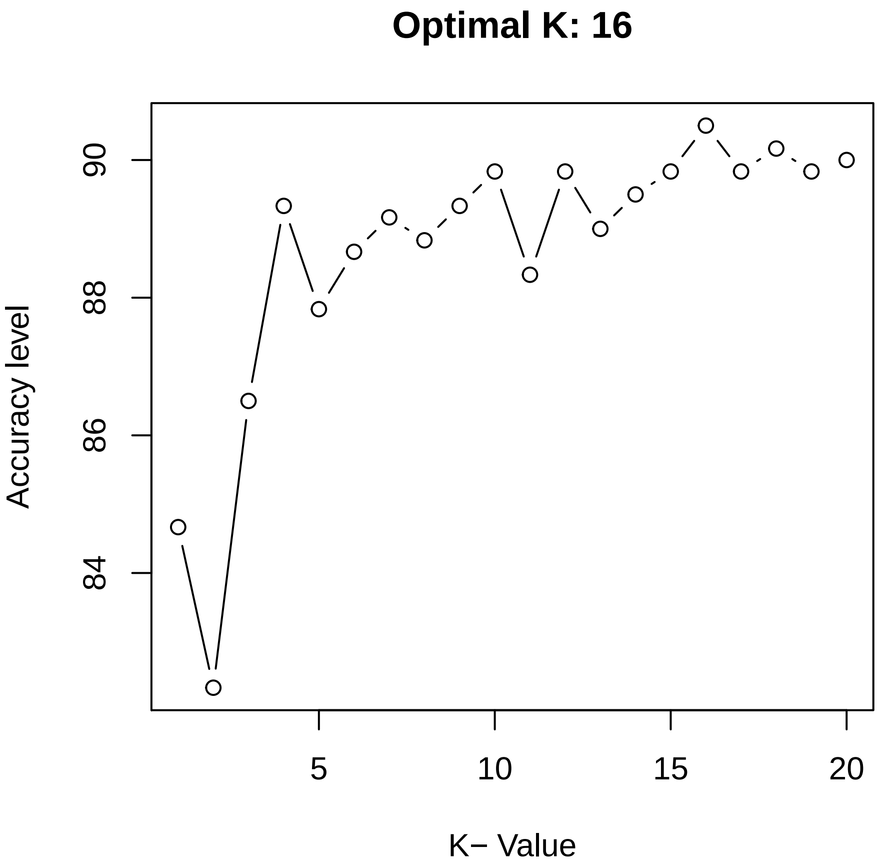

Extent vs Class

## Plots 2

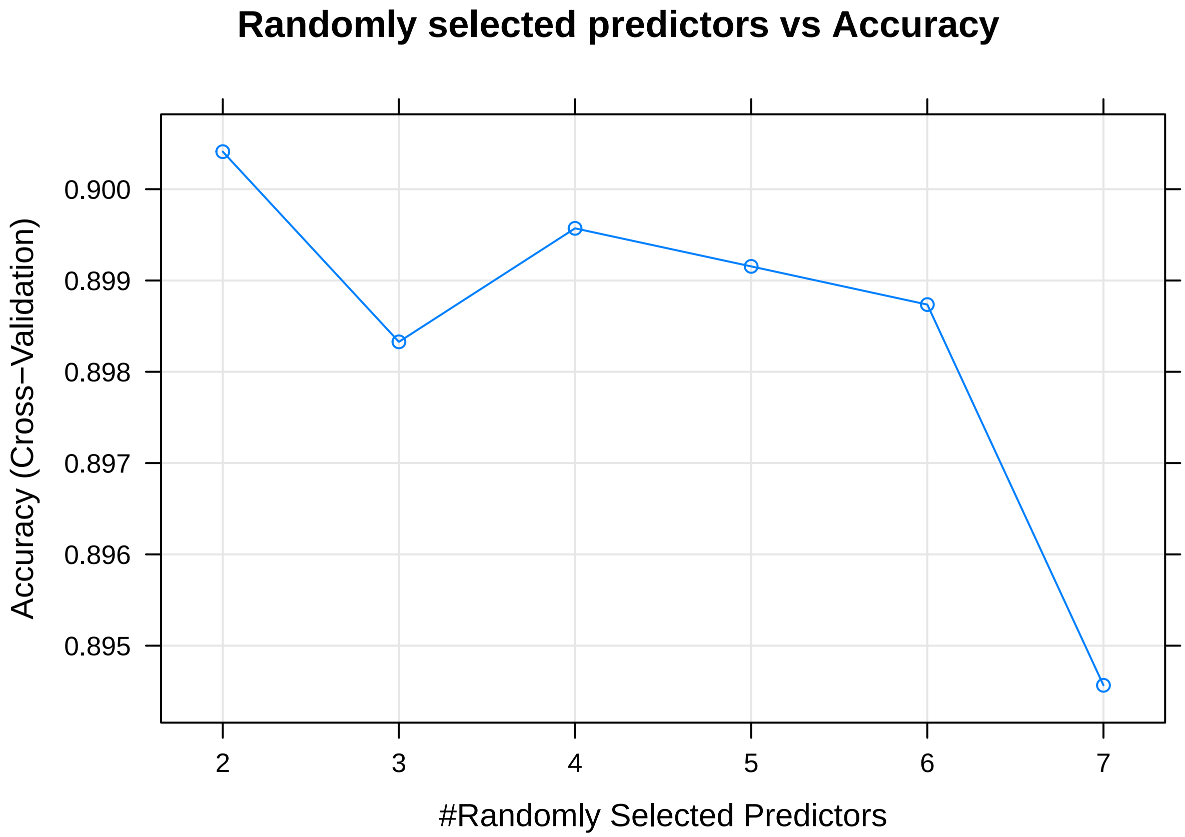


Perimeter
2000
1500
1000
500
50000
100000
150000
200000
250000
Area
as.factor(Class)
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
MajorAxisLength
600
400
200
0e+00
1e+05
2e+05
ConvexArea
as.factor(Class)
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
Perimeter
2000
1500
1000
500
0.4
0.6
0.8
Eccentricity
as.factor(Class)
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
Perimeter
2000
1500
1000
500
200
400
600
MajorAxisLength
as.factor(Class)
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
Perimeter
2000
1500
1000
500
0.60
0.65
0.70
0.75
0.80
0.85
Extent
as.factor(Class)
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
ConvexArea
2e+05
1e+05
0e+00
0.4
0.6
0.8
Eccentricity
as.factor(Class)
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA

```
## Figure 1.

##   BOMBAY CALI DERMASON HOROZ SEKER SIRA
## 1      0    0        1     0     0    0
## 2      0    1        0     0     0    0
## 3      0    0        0     0     1    0
## 4      0    0        0     0     0    1
## 5      0    1        0     0     0    0
## 6      0    0        0     1     0    0

## hidden: 15, 10, 3    thresh: 0.01    rep: 1/1    steps:   24901  error: 87.00443 time: 3.39 mins

## hidden: 15, 10, 3    thresh: 0.01    rep: 1/1    steps:   22702  error: 128.50683    time: 3.75 mins
## Figure 2
```

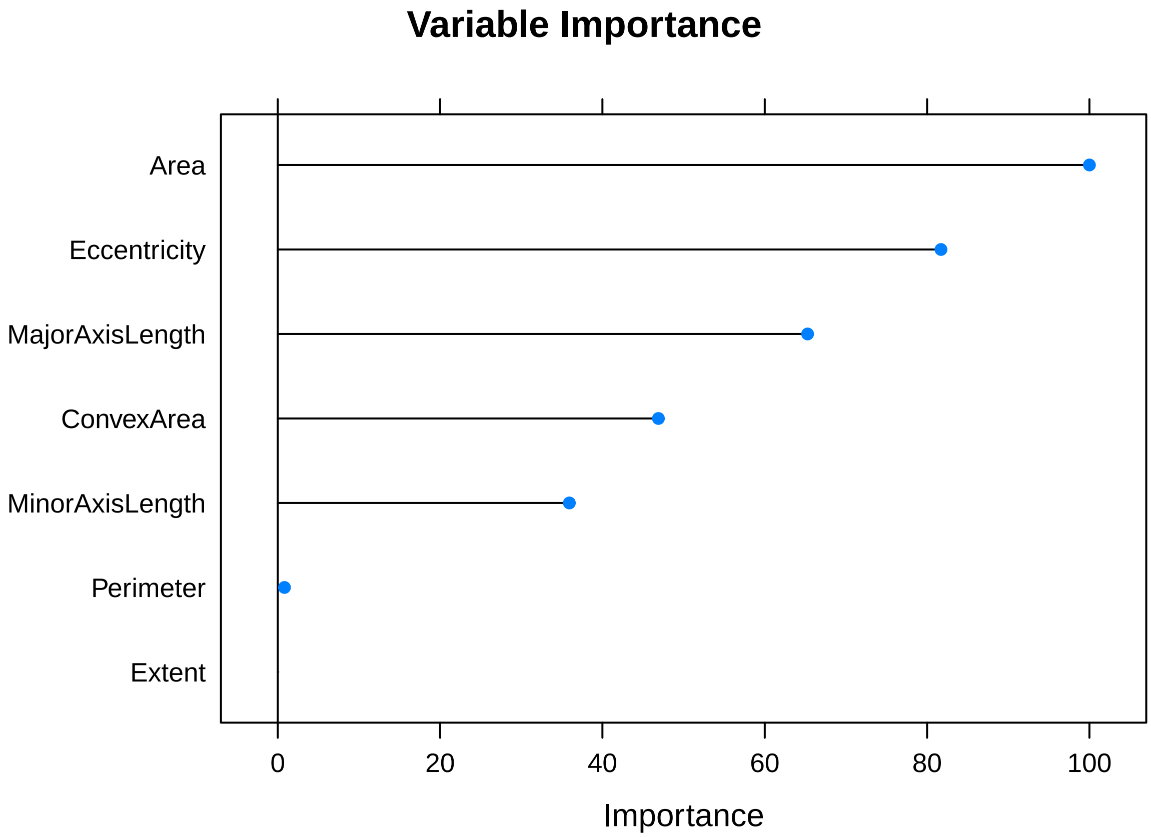


```
## note: only 6 unique complexity parameters in default grid. Truncating the grid to 6 .

## note: only 6 unique complexity parameters in default grid. Truncating the grid to 6 .
## Figure 3
```



mtry	Accuracy	Kappa	AccuracySD	KappaSD
2	0.9004114	0.8804735	0.0234081	0.0280973
3	0.8983281	0.8779800	0.0242966	0.0291651
4	0.8995712	0.8794702	0.0244285	0.0293253
5	0.8991545	0.8789706	0.0228269	0.0273988
6	0.8987361	0.8784658	0.0236936	0.0284428
7	0.8945659	0.8734561	0.0296668	0.0356095

```
## Figure 4
```


Variable Importance
Area
Eccentricity
MajorAxisLength
ConvexArea
MinorAxisLength
Perimeter
Extent
0
20
40
60
80
100
Importance

Model Comparision

Table 3: Test Model Metrics

	NNET	LDA	KNN	SVM	RF
Test Accuracy	0.8850	0.8617	0.9017	0.8900	0.8933
Actual Price	4.5052	4.5052	4.1561	4.5052	4.5052
Predicted Price	4.5164	4.5144	4.1522	4.5190	4.5487
F1 Score	0.8828	0.8615	0.9044	0.8892	0.8927

Table 4: Precision by Bean

	BOMBAY	CALI	DERMASON	HOROZ	SEKER	SIRA
NNET	1	0.9278	0.8257	0.8710	0.8738	0.7907
LDA	1	0.8763	0.7890	0.8387	0.8447	0.8023
KNN	1	0.9457	0.8667	0.8966	0.9369	0.7838
SVM	1	0.9072	0.7890	0.8710	0.9126	0.8488
RF	1	0.9381	0.8257	0.8602	0.8835	0.8372

Sample A Predictions

Table 5: Prediction for Sample A

	NNET	LDA	KNN	SVM	RF
Predicted Price	4.9453	4.5269	4.706	4.5893	4.6118
Predicted Weight	1.0133	0.9794	0.9884	0.9892	0.9899
Price Accuracy Prediction	0.9875	0.9788	0.9885	0.989	0.9898
Error in Dollars	+/- 0.06	+/- 0.1	+/- 0.05	+/- 0.05	+/- 0.05

Table 6: Predicted Bean Counts Sample A

Model	Bombay	Cali	Dermason	Horoz	Seker	Sira
NNET	61	326	201	14	127	47
LDA	22	343	22	17	331	41
KNN	46	320	148	34	173	55
SVM	22	356	14	13	342	29
RF	22	360	14	11	339	30

Sample B Predictions

Table 7: Prediction for Sample B

	NNET	LDA	KNN	SVM	RF
Predicted Price	10.9067	3.32	9.7644	3.2283	3.1318
Predicted Weight	2.2221	1.0431	1.9666	1.0396	1.0305
Price Accuracy Prediction	0.4784	0.9396	0.5392	0.9429	0.9546
Error in Dollars	+/- 5.69	+/- 0.2	+/- 4.5	+/- 0.18	+/- 0.14

Table 8: Predicted Bean Counts Sample B

Model	Bombay	Cali	Dermason	Horoz	Seker	Sira
NNET	259	345	296	207	78	188
LDA	0	2	761	14	228	368
KNN	168	457	256	123	135	234
SVM	0	0	782	10	242	339
RF	0	0	815	12	232	314

Sample C Predictions

Table 9: Prediction for Sample C

	NNET	LDA	KNN	SVM	RF
Predicted Price	7.7195	3.377	6.9454	3.3117	3.2912
Predicted Weight	1.5392	1.0049	1.4243	1.0074	1.0063
Price Accuracy Prediction	0.6749	0.9932	0.7156	0.9896	0.9911
Error in Dollars	+/- 2.51	+/- 0.02	+/- 1.97	+/- 0.03	+/- 0.03

Table 10: Predicted Bean Counts Sample C

Model	Bombay	Cali	Dermason	Horoz	Seker	Sira
NNET	145	391	155	102	118	71
LDA	0	115	171	519	6	171
KNN	101	398	106	128	168	81
SVM	0	105	168	539	9	161
RF	0	98	164	544	9	167

Sources:

1. https://www.rdocumentation.org/packages/neuralnet/versions/1.44.2/topics/neuralnet
2. https://www.edureka.co/blog/knn-algorithm-in-r/
3. https://rstudio-pubs-static.s3.amazonaws.com/316172_a857ca788d1441f8be1bcd1e31f0e875.html
4. https://www.edureka.co/blog/random-forest-classifier/
5. https://www.blopig.com/blog/2017/04/a-very-basic-introduction-to-random-forests-using-r/
6. https://github.com/mariocastro73/ML2020-2021/blob/master/scripts/caret-rf.R
7. KOKLU, M. and OZKAN, I.A., (2020), "Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques." Computers and Electronics in Agriculture, 174, 105507.
8. Saunders, Christopher.(2018).Final LDA Example. https://d2l.sdbor.edu/d2l/le/content/1543761/viewContent/8840088/View
9. https://www.edureka.co/blog/support-vector-machine-in-r/
10. https://www.r-project.org/conferences/useR-2013/Tutorials/kuhn/user_caret_2up.pdf
11. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2013). An introduction to statistical learning : with applications in R. New York :Springer