Title: “IRIS flowers Project” || Author: “Amit Ranjan”

Introduction

We are going to use the iris flowers dataset. This dataset is famous because it is used as the “hello world” dataset in machine learning and statistics by pretty much everyone. 1) The dataset contains 150 observations of iris flowers. There are four columns of measurements of the flowers in centimeters. 2) The fifth column is the species of the flower observed. All observed flowers belong to one of three species. 3) You can learn more about this dataset on Wikipedia. [https://en.wikipedia.org/wiki/Iris_flower_data_set (https://en.wikipedia.org/wiki/Iris_flower_data_set)]

Here is what we are going to do in this step:

1. Load the iris data the easy way.
2. Load the iris data from CSV (optional, for purists).
3. Separate the data into a training dataset and a validation dataset.

Our output will be 2 different files:

1. A Report and R Code in RMD file format that generates our recommendations 2 A report in PDF format, which we will obtain by publishing this R Code in HTML and printing as PDF

Install the packages we are going to use today. Packages are third party add-ons or libraries that we can use in R.

Install the packages we are going to use today. Packages are third party add-ons or libraries that we can use in R.

```
# Note: this process could take a couple of minutes to download and install all librarie
s (total download around 100+ MB)
if(!require(readr)) install.packages("readr", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(tidyr)) install.packages("tidyr", repos = "http://cran.us.r-project.org")
if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.org"
)
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org"
)
if(!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.us.r-project.
org")
if(!require(dslabs)) install.packages("dslabs", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-projec
t.org")
if(!require(ggrepel)) install.packages("ggrepel", repos = "http://cran.us.r-project.org"
)
if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.or
g")
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.
org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(ellipse)) install.packages("ellipse", repos = "http://cran.us.r-project.org"
)
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-pr
oject.org")
if(!require(kernlab)) install.packages("kernlab", repos = "http://cran.us.r-project.org"
)


library(randomForest)
library(kernlab)
library(ellipse)
library(readr)
library(dplyr)
library(tidyr)
library(stringr)
library(ggplot2)
library(gridExtra)
library(dslabs)
library(data.table)
library(ggrepel)
library(ggthemes)
library(tidyverse)
library(caret)
# https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
```

Load The Data and perform Exploratory Data Analysis

This section repeart the data explorartion with the answers to the quizz online, and then we will verify some of the data quality which we learnt during class followed by additional data analysis.

```
# attach the iris dataset to the environment
data(iris)
# rename the dataset
dataset <- iris
#https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
```

Create a Validation Dataset

We need to know that the model we created is any good.Later, we will use statistical methods to estimate the accuracy of the models that we create on unseen data. We also want a more concrete estimate of the accuracy of the best model on unseen data by evaluating it on actual unseen data. That is, we are going to hold back some data that the algorithms will not get to see and we will use this data to get a second and independent idea of how accurate the best model might actually be. We will split the loaded dataset into two, 80% of which we will use to train our models and 20% that we will hold back as a validation dataset.

Create a list of 80% of the rows in the original dataset we can use for training

```
validation_index <- createDataPartition(dataset$Species, p=0.80, list=FALSE)
# select 20% of the data for validation
validation <- dataset[-validation_index,]
# use the remaining 80% of data to training and testing the models
dataset <- dataset[validation_index,]
```

Summarize Dataset

Now it is time to take a look at the data.

In this step we are going to take a look at the data a few different ways:

1. Dimensions of the dataset.
2. Types of the attributes.
3. Peek at the data itself.
4. Levels of the class attribute.
5. Breakdown of the instances in each class.
6. Statistical summary of all attributes.

Any Null of dataset

```
anyNA(dataset)
```

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

Dimensions of dataset

```
dim(dataset)
```

```
## [1] 120   5
```

List types for each attribute

```
sapply(dataset, class)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##    "numeric"    "numeric"    "numeric"    "numeric"     "factor"
```

Take a peek at the first 5 rows of the data

```
head(dataset)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
## 7          4.6         3.4          1.4         0.3  setosa
## 9          4.4         2.9          1.4         0.2  setosa
```

List the levels for the class

```
levels(dataset$Species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

Summarize the class distribution

```
percentage <- prop.table(table(dataset$Species)) * 100
cbind(freq=table(dataset$Species), percentage=percentage)
```

```
##            freq percentage
## setosa       40   33.33333
## versicolor   40   33.33333
## virginica    40   33.33333
```

Summarize attribute distributions

```
summary(dataset)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.200   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.450   Median :1.400
##  Mean   :5.867   Mean   :3.055   Mean   :3.797   Mean   :1.198
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.200   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :40
##  versicolor:40
##  virginica :40
##
##
##
```

Visualize Dataset

We need to extend that with some visualizations.

We are going to look at two types of plots: 1) Univariate plots to better understand each attribute. 2) Multivariate plots to better understand the relationships between attributes.

It is helpful with visualization to have a way to refer to just the input attributes and just the output attributes. Let's set that up and call the inputs attributes x and the output attribute (or class) y.

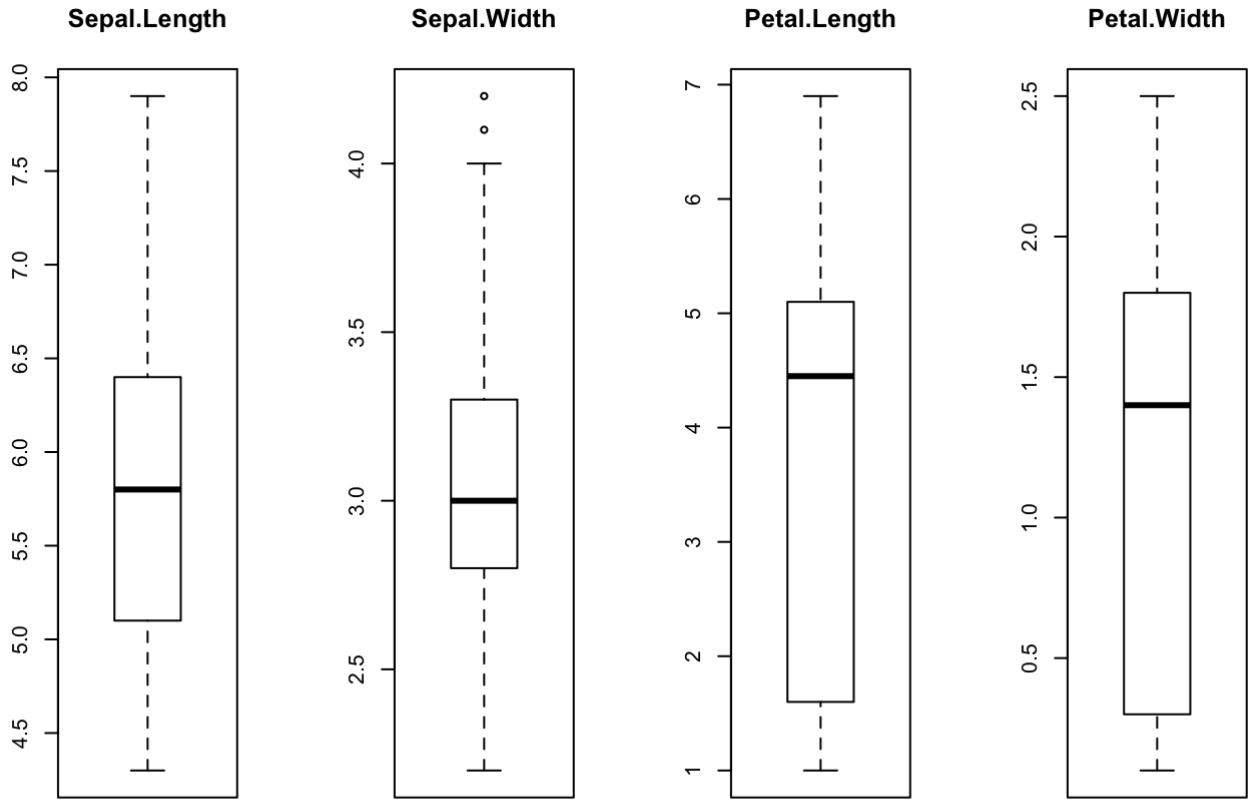
Split input and output

```
x <- dataset[,1:4]
y <- dataset[,5]
```

Given that the input variables are numeric, we can create box and whisker plots of each.

Boxplot for each attribute on one image

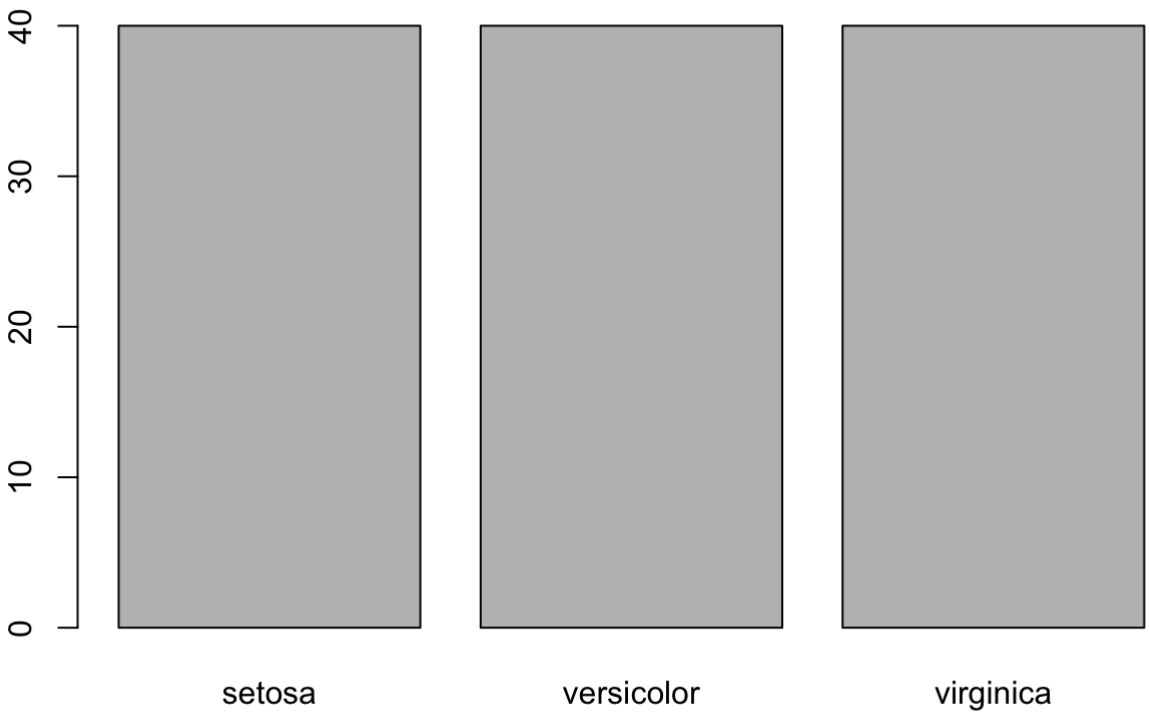
```
par(mfrow=c(1,4))
  for(i in 1:4) {
  boxplot(x[,i], main=names(iris)[i])
  }
```



We can also create a barplot of the Species class variable to get a graphical representation of the class distribution (generally uninteresting in this case because they're even). This confirms what we learned in the last section, that the instances are evenly distributed across the three class:

Barplot for class breakdown

```
plot(y)
```



##

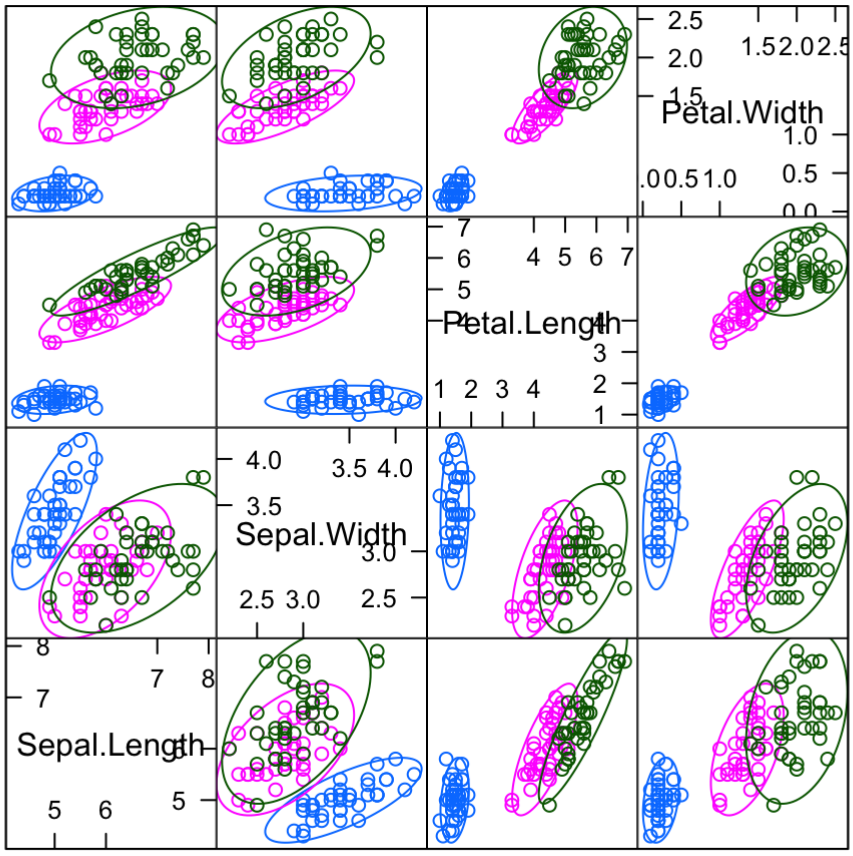
Multivariate Plots

Now we can look at the interactions between the variables.

First let's look at scatterplots of all pairs of attributes and color the points by class. In addition, because the scatterplots show that points for each class are generally separate, we can draw ellipses around them.

scatterplot matrix showing various correlations and overlap between different data. ### Scatterplot matrix

```
featurePlot(x=x, y=y, plot="ellipse")
```



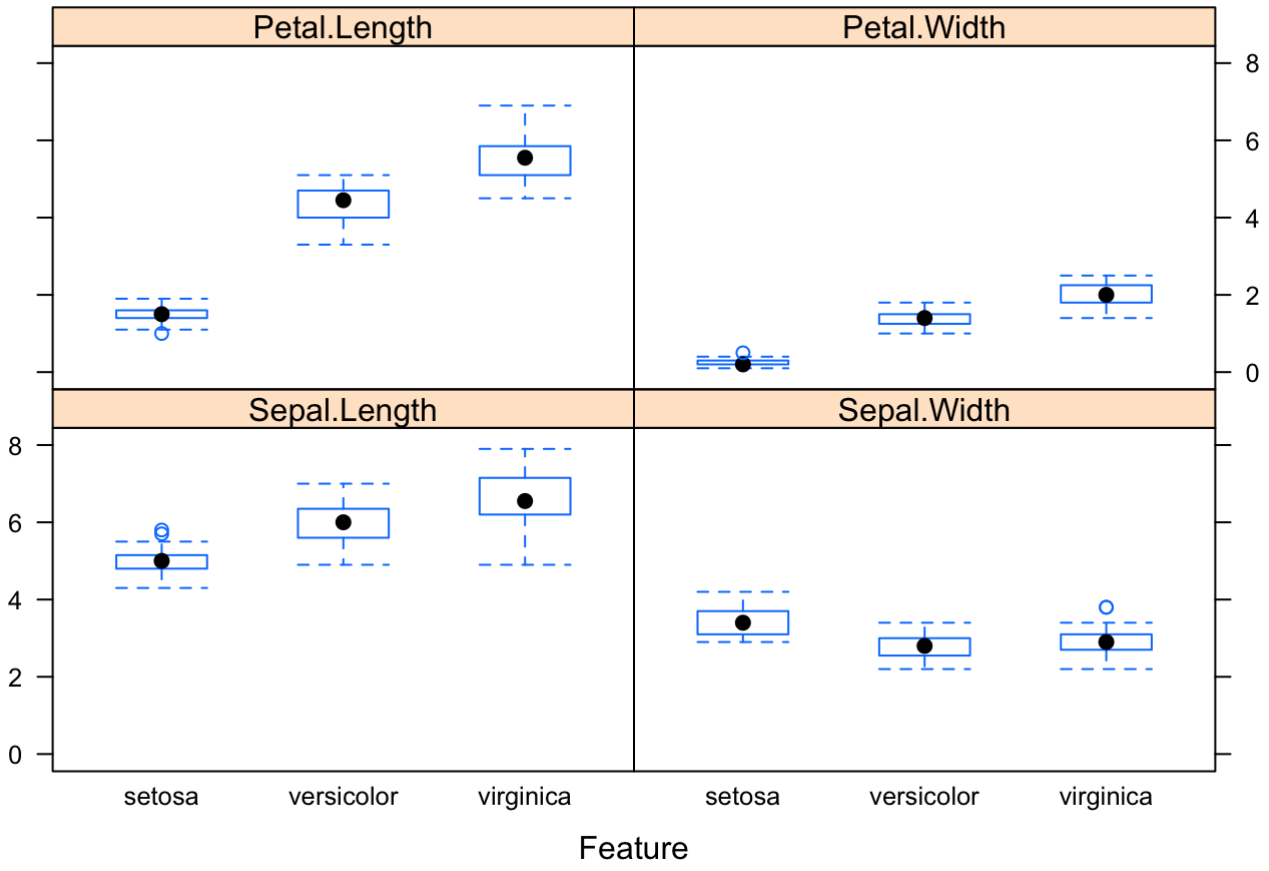
Scatter Plot Matrix

We can also look at box and whisker plots of each input variable again, but this time broken down into separate plots for each class. This can help to tease out obvious linear separations between the classes. This is useful to see that there are clearly different distributions of the attributes for each class value.

Like he boxplots, we can see the difference in distribution of each attribute by class value. We can also see the Gaussian-like distribution (bell curve) of each attribute.

Box and whisker plots for each attribute

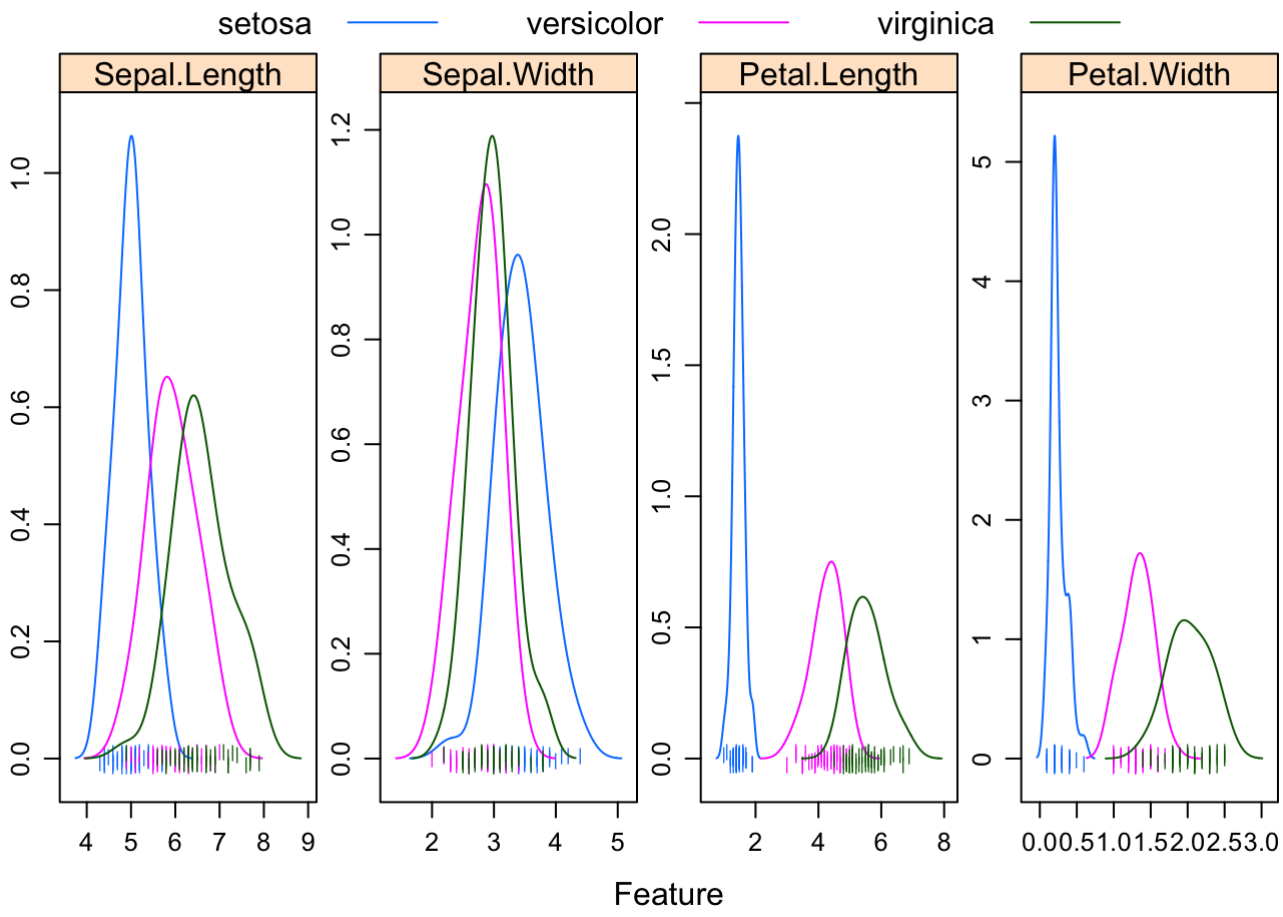
```
featurePlot(x=x, y=y, plot="box")
```



Density plots for each attribute by class value helps understand how each attribute lies gainat each other and vs the different species. We can see setos has larger attribute values for most attributes.

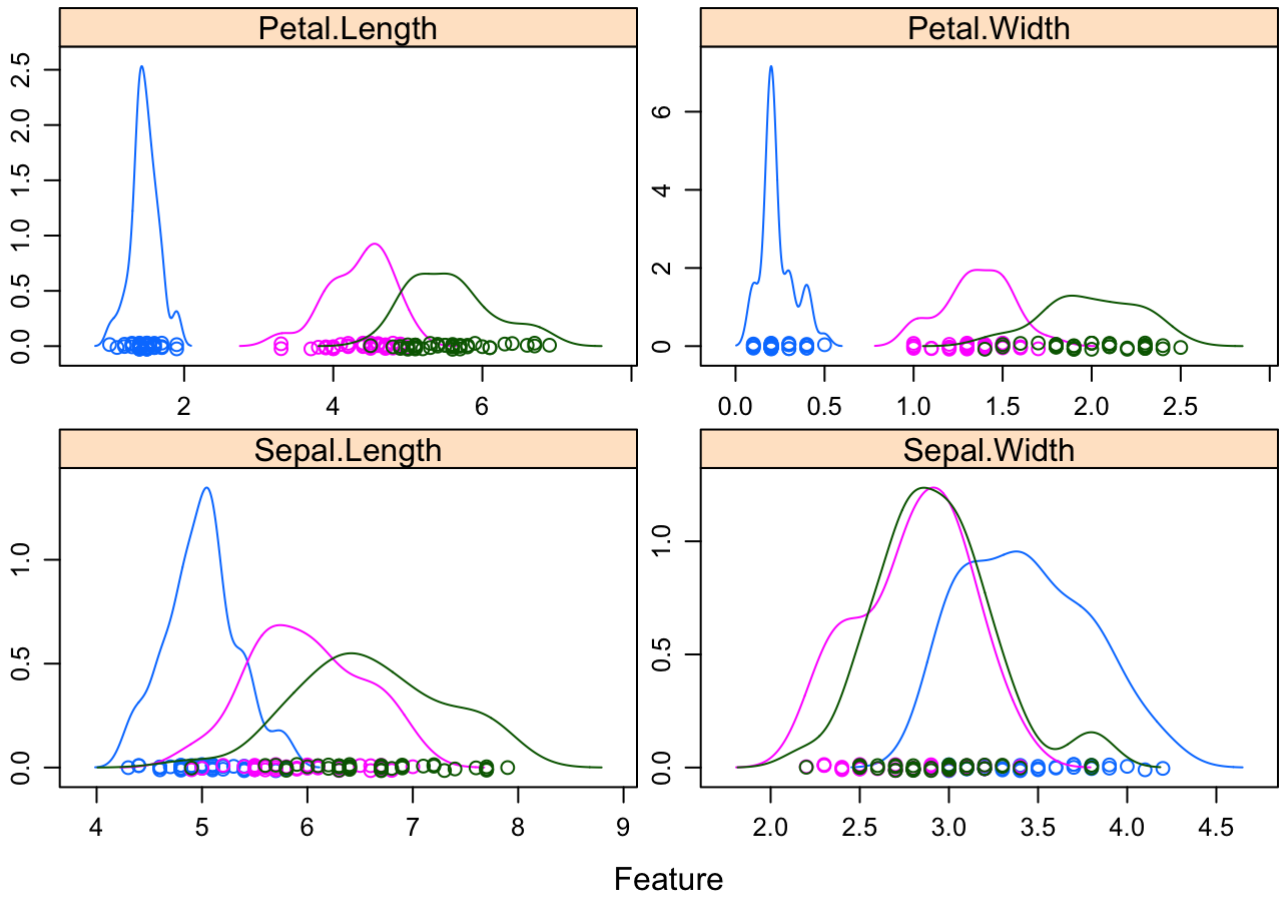
Density plots for each attribute by class value

```
featurePlot(x = iris[, 1:4],
            y = iris$Species,
            plot = "density",
            ## Pass in options to xyplot() to
            ## make it prettier
            scales = list(x = list(relation="free"),
                          y = list(relation="free")),
            adjust = 1.5,
            pch = "|",
            layout = c(4, 1),
            auto.key = list(columns = 3))
```



Density plots for each attribute by class value

```
scales <- list(x=list(relation="free"), y=list(relation="free"))
featurePlot(x=x, y=y, plot="density", scales=scales)
```



Evaluate Some Algorithms

Now it is time to create some models of the data and estimate their accuracy on unseen data.

1. Set-up the test harness to use 10-fold cross validation.
2. Build 5 different models to predict species from flower measurements
3. Select the best model.

Test Harness

We will 10-fold crossvalidation to estimate accuracy.

This will split our dataset into 10 parts, train in 9 and test on 1 and release for all combinations of train-test splits. We will also repeat the process 3 times for each algorithm with different splits of the data into 10 groups, in an effort to get a more accurate estimate.

Run algorithms using 10-fold cross validation

```
control <- trainControl(method="cv", number=10)
metric <- "Accuracy"
```

Build Models

We don't know which algorithms would be good on this problem or what configurations to use. We get an idea from the plots that some of the classes are partially linearly separable in some dimensions, so we are expecting generally good results.

###Let's evaluate 5 different algorithms:

1. Linear Discriminant Analysis (LDA)
2. Classification and Regression Trees (CART).
3. k-Nearest Neighbors (kNN).
4. Support Vector Machines (SVM) with a linear kernel.
5. Random Forest (RF)
6. Weighted k-Nearest Neighbors Classification (KKNN)
7. Naive Bayes

This is a good mixture of simple linear (LDA), nonlinear (CART, kNN) and complex nonlinear methods (SVM, RF). We reset the random number seed before reach run to ensure that the evaluation of each algorithm is performed using exactly the same data splits. It ensures the results are directly comparable.

Train for following models linear, nonlinear, KNN, SVM, Random Forest, naive_bayes, kknn

```
# a) linear algorithms
set.seed(7)
fit.lda <- train(Species~., data=dataset, method="lda", metric=metric, trControl=contro
l)
# b) nonlinear algorithms
# CART
set.seed(7)
fit.cart <- train(Species~., data=dataset, method="rpart", metric=metric, trControl=cont
rol)
# kNN
set.seed(7)
fit.knn <- train(Species~., data=dataset, method="knn", metric=metric, trControl=contro
l)
# c) advanced algorithms
# SVM
set.seed(7)
fit.svm <- train(Species~., data=dataset, method="svmRadial", metric=metric, trControl=c
ontrol)
# Random Forest
set.seed(7)
fit.rf <- train(Species~., data=dataset, method="rf", metric=metric, trControl=control)

# kknn
set.seed(7)
fit.kknn <- train(Species~., data=dataset, method="kknn", metric=metric, trControl=contr
ol)

# naive_bayes
set.seed(7)
fit.naive_bayes <- train(Species~., data=dataset, method="naive_bayes", metric=metric, t
rControl=control)
```

Select Best Model

We now have 5 models and accuracy estimations for each. We need to compare the models to each other and select the most accurate.

We can report on the accuracy of each model by first creating a list of the created models and using the summary function.

Summarize accuracy of models

```
results <- resamples(list(lda=fit.lda, cart=fit.cart, knn=fit.knn, svm=fit.svm, rf=fit.r
f, kknn = fit.kknn, naive_bayes= fit.naive_bayes))
summary(results)
```

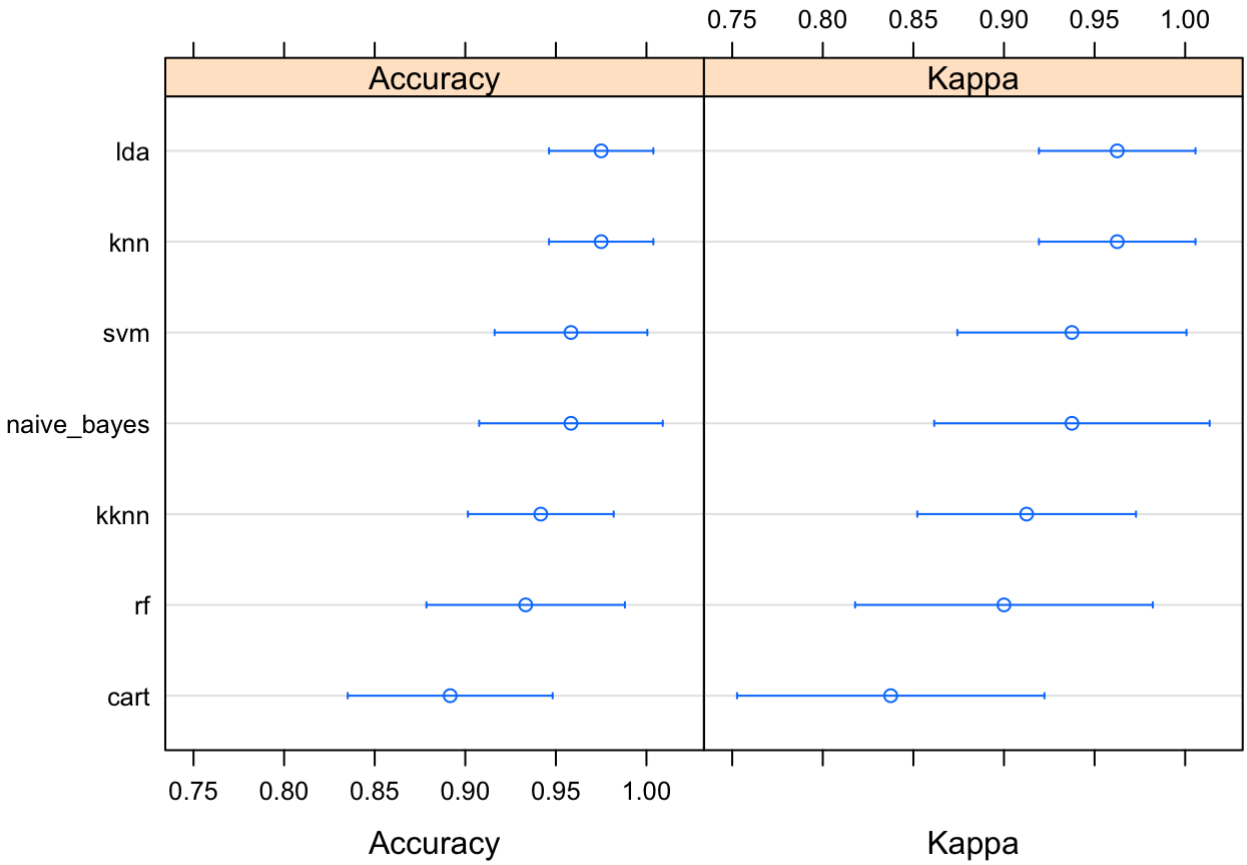
```
##
## Call:
## summary.resamples(object = results)
##
## Models: lda, cart, knn, svm, rf, kknn, naive_bayes
## Number of resamples: 10
##
## Accuracy
##                  Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
## lda         0.9166667 0.9375000 1.0000000 0.9750000 1.0000000    1    0
## cart        0.7500000 0.8333333 0.9166667 0.8916667 0.9166667    1    0
## knn         0.9166667 0.9375000 1.0000000 0.9750000 1.0000000    1    0
## svm         0.8333333 0.9166667 1.0000000 0.9583333 1.0000000    1    0
## rf          0.8333333 0.8541667 0.9583333 0.9333333 1.0000000    1    0
## kknn        0.8333333 0.9166667 0.9166667 0.9416667 1.0000000    1    0
## naive_bayes 0.8333333 0.9375000 1.0000000 0.9583333 1.0000000    1    0
##
## Kappa
##                Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
## lda         0.875 0.90625 1.0000 0.9625   1.000    1    0
## cart        0.625 0.75000 0.8750 0.8375   0.875    1    0
## knn         0.875 0.90625 1.0000 0.9625   1.000    1    0
## svm         0.750 0.87500 1.0000 0.9375   1.000    1    0
## rf          0.750 0.78125 0.9375 0.9000   1.000    1    0
## kknn        0.750 0.87500 0.8750 0.9125   1.000    1    0
## naive_bayes 0.750 0.90625 1.0000 0.9375   1.000    1    0
```

We can see the accuracy of each classifier and also other metrics like Kappa:

We can also create a plot of the model evaluation results and compare the spread and the mean accuracy of each model. There is a population of accuracy measures for each algorithm because each algorithm was evaluated 10 times (10 fold cross validation).

compare accuracy of models

```
dotplot(results)
```



This gives a nice summary of what was used to train the model and the mean and standard deviation (SD) accuracy achieved, specifically 97.5% accuracy +/- 4%

Summarize Best Model

```
print(fit.lda)
```

```
## Linear Discriminant Analysis
##
## 120 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 108, 108, 108, 108, 108, 108, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.975     0.9625
```

Make Predictions

The LDA was the most accurate model. Now we want to get an idea of the accuracy of the model on our validation set.

This will give us an independent final check on the accuracy of the best model. It is valuable to keep a validation set just in case you made a slip during such as overfitting to the training set or a data leak. Both will result in an overly optimistic result.

We can run the LDA model directly on the validation set and summarize the results in a confusion matrix.

Estimate skill of LDA on the validation dataset

```
predictions <- predict(fit.lda, validation)
confusionMatrix(predictions, validation$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         0
##   virginica       0          0        10
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.8843, 1)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 4.857e-15
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           1.0000
## Specificity                 1.0000            1.0000           1.0000
## Pos Pred Value              1.0000            1.0000           1.0000
## Neg Pred Value              1.0000            1.0000           1.0000
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3333           0.3333
## Detection Prevalence        0.3333            0.3333           0.3333
## Balanced Accuracy           1.0000            1.0000           1.0000
```

Additional Analysis and obersvation

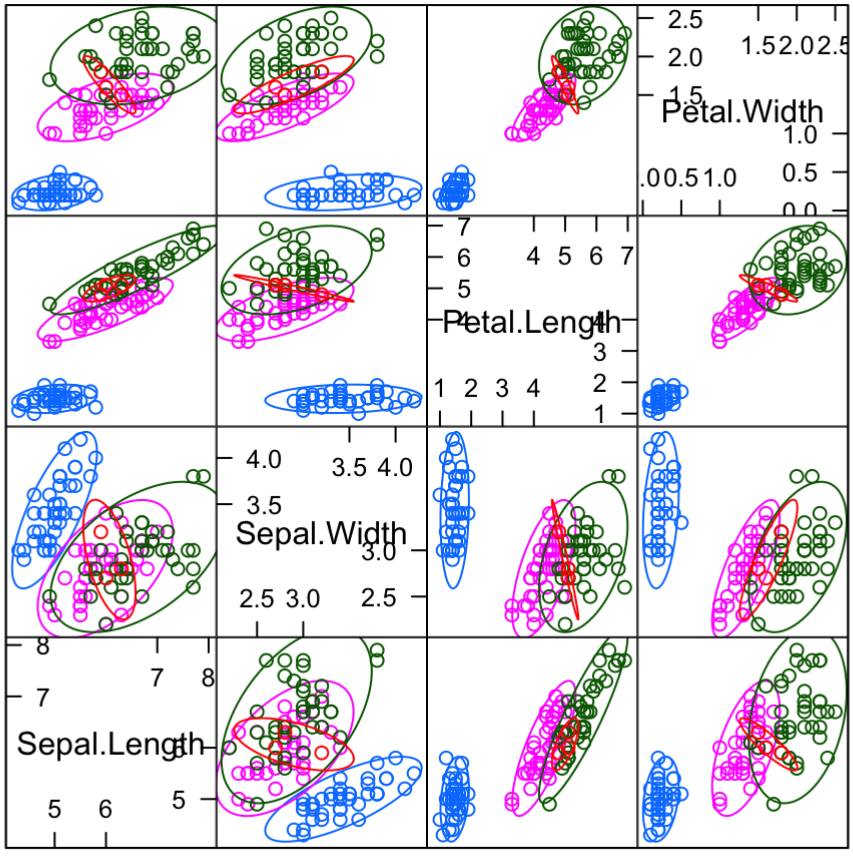
NEW: Evaluating performance on the training dataset, which has 80% of records code above showed that linear discriminant analysis performed best, but what we can we learn about the incorrectly predicted flowers?

Assign former “dataset” to trainset to make clear its identity

```
trainset <- dataset
trainset$predictionsT <- predict(fit.lda, trainset)
trainset$y2 <- ifelse (trainset$Species == trainset$predictionsT, trainset$Species, "mis
match")
# split input and output
x2 <- trainset[,1:4]
# y has to be a factor or featurePlot crashes (returns NULL)
y2 <- as.factor(trainset$y2)
# report details about mismatches
trainset[trainset$y2=="mismatch",]
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 71           5.9         3.2          4.8         1.8 versicolor
## 84           6.0         2.7          5.1         1.6 versicolor
## 134          6.3         2.8          5.1         1.5  virginica
##     predictionsT       y2
## 71     virginica mismatch
## 84     virginica mismatch
## 134   versicolor mismatch
```

```
# Scatterplot matrix with a the mismatches in red.
#  As expected, the mismatches appear in the intersection of
#   green (species "virginica") and pink (species "versicolor")
featurePlot(x=x2, y=y2, plot="ellipse")
```



Scatter Plot Matrix

Scatterplot matrix with a the mismatches in red. As expected, the mismatches appear in the intersection of green (species “virginica”) and pink (species “versicolor”)

Conclusion

As stated earlier, I tested Naive Bayes, Random Forest, SVD, Recommenderlab, KNN, Kmeans, and various other models and algorithms. Some were fast but the accuracy poor. Others were accurate on smaller sets (Random Forest) offered no reliable means to split and combine.

In conclusion, utilizing only 3 elements of the data set (sepeal and petal leangth and width) we were able to predict. Finding the right algorithm that suits the existing data set is often more useful than exploring augmentation of the data to reach your own conclusions, such as including or introducing an artificial bias or weights.

As we have seen above out of 7 algorithms, the LDA performed best. At the end Evaluating performance on the training dataset, which has 80% of records code above showed that linear discriminant analysis performed best, but what we can we learn about the incorrectly predicted flowers Scatterplot matrix with a the mismatches in red. As expected, the mismatches appear in the intersection of green (species “virginica”) and pink (species “versicolor”)