Machine Learning in R

Chenshu Liu

April 2022

Contents

Packages

```
# for data splitting into training and testing (general data manipulation)
library(caTools)

# dataset for logistic regression
# install.packages("mlbench")
library(mlbench)

# Decision tree
# install.packages("FSelector")
# install.packages("caret", dependencies = T)
# install.packages("rpart.plot")
# install.packages("data.tree")
# install.packages("rJava")

# for constructing decision tree
system("java -version")
library(FSelector)
library(rpart)
# for test train set split
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.2
```

```
library(dplyr)
# plotting the decision tree
library(rpart.plot)
library(data.tree)

# Random forest
# install.packages("randomForest")
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.1.2
```

```
# Support Vector Machine
library(e1071)
```

Linear Regression

Linear regression is a way to find the best fit linear expression that can show the observed trend(s). The parameters of the best fit line $y = mx + c$ can be calculated by:

$$m = \frac{(n \times \Sigma(x \times y)) - (\Sigma(x) \times \Sigma(y))}{(n \times \Sigma(x^2)) - (\Sigma(x)^2)}$$

$$c = \frac{(\Sigma(y) \times \Sigma(x^2)) - (\Sigma(x) \times \Sigma(x \times y))}{(n \times \Sigma(x^2)) - (\Sigma(x)^2)}$$

Data

```
sales <- read.csv("/Users/chenshu/Documents/Programming/R/Machine Learning with R/datasets/revenue.csv")

# split into training and testing data
set.seed(2)
# SplitRatio means the percentage of data for training
split <- caTools::sample.split(sales$Profit, SplitRatio = 0.7)
train <- sales[split, ]
test <- sales[!split, ]
```

Modeling

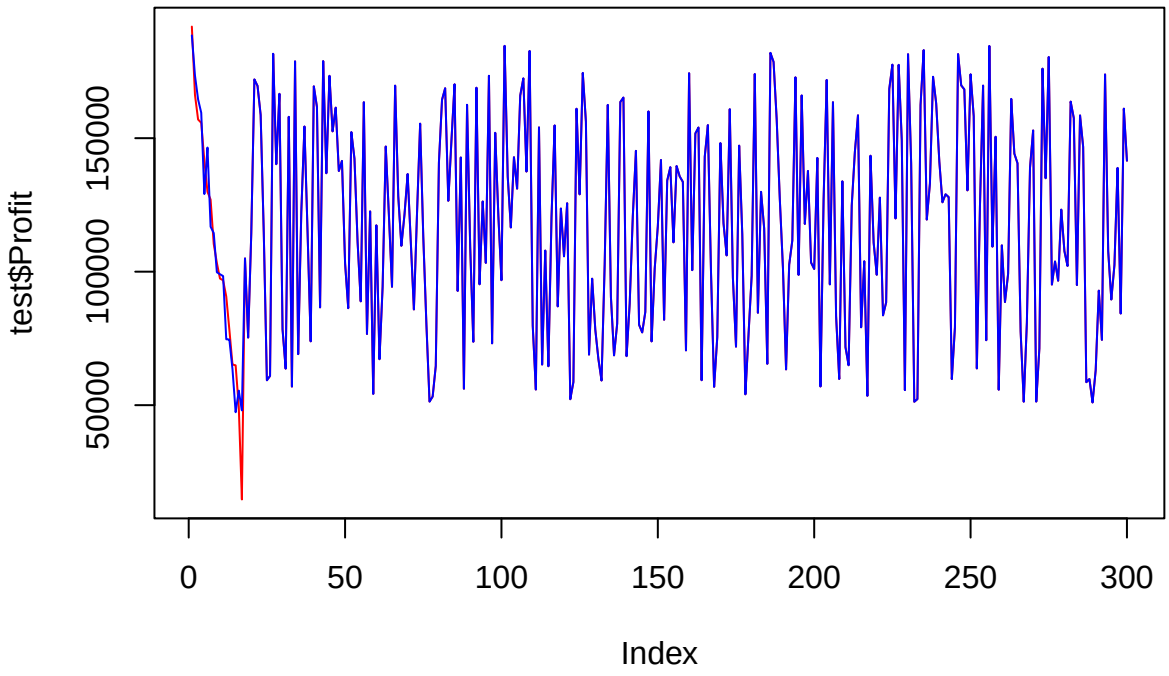
```
Model <- lm(Profit ~., data = train)
summary(Model)
```

```
##
## Call:
## lm(formula = Profit ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16134.9    -32.2    -17.7     -6.5  10133.9
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.395e+04  1.145e+03  47.126  < 2e-16 ***
## Paid         8.155e-01  6.982e-03 116.802  < 2e-16 ***
## Organic     -6.007e-02  9.547e-03  -6.292 5.53e-10 ***
## Social       2.488e-02  3.247e-03   7.661 6.21e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1480 on 696 degrees of freedom
## Multiple R-squared:  0.9986, Adjusted R-squared:  0.9986
## F-statistic: 1.693e+05 on 3 and 696 DF,  p-value: < 2.2e-16
```

Predict

```r
pred <- predict(Model, test)

# comparing predicted vs. actual values
plot(test$Profit, type = 'l', lty = 1.8, col = "red")
lines(pred, type = 'l', lty = 1.8, col = "blue")
```




```r
# determining prediction accuracy
rmse <- sqrt(mean(pred - test$Profit)^2)
rmse
```

```
## [1] 61.56887
```

Logistic Regression

Logistic regression is a **classification algorithm**, not a linear prediction algorithm. Different from linear regression, which is usually used to determine the magnitude of the effect, logistic regression is used to predict binary outcome.

Data

```
data(PimaIndiansDiabetes)
log_df <- PimaIndiansDiabetes
head(log_df)
```

```
##   pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 1        6     148       72      35       0 33.6    0.627  50      pos
## 2        1      85       66      29       0 26.6    0.351  31      neg
## 3        8     183       64       0       0 23.3    0.672  32      pos
## 4        1      89       66      23      94 28.1    0.167  21      neg
## 5        0     137       40      35     168 43.1    2.288  33      pos
## 6        5     116       74       0       0 25.6    0.201  30      neg
```

```
# splitting data
split <- caTools::sample.split(log_df$diabetes, SplitRatio = 0.7)
train <- log_df[split, ]
test <- log_df[!split, ]

# data pre-processing
# logistic regression takes in factor type variables
train$diabetes <- as.factor(train$diabetes)
```

Modeling

```
log_mod <- glm(diabetes ~., data = train, family = "binomial")
summary(log_mod)
```

```
##
## Call:
## glm(formula = diabetes ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4160  -0.7396  -0.4437   0.7655   2.7580
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -7.9513478  0.8219838  -9.673  < 2e-16 ***
## pregnant     0.1365115  0.0372059   3.669 0.000243 ***
## glucose      0.0313336  0.0041092   7.625 2.44e-14 ***
## pressure    -0.0127133  0.0061062  -2.082 0.037339 *
## triceps      0.0002463  0.0080616   0.031 0.975628
```

```
## insulin     -0.0010894  0.0010684  -1.020 0.307890
## mass         0.0894903  0.0177557   5.040 4.65e-07 ***
## pedigree     0.7669301  0.3433048   2.234 0.025486 *
## age          0.0152128  0.0108031   1.408 0.159073
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 696.28  on 537  degrees of freedom
## Residual deviance: 520.90  on 529  degrees of freedom
## AIC: 538.9
##
## Number of Fisher Scoring iterations: 5
```

Prediction

```
pred <- predict(log_mod, test, type = "response")

# confusion matrix to check prediction accuracy
table(Actual_value = test$diabetes, Predicted_value = pred > 0.5)
```

```
##             Predicted_value
## Actual_value FALSE TRUE
##          neg   137   13
##          pos    30   50
```

Decision Tree

1. Decision tree is a tree shape algorithm that is used to determine a course of actions, with each branch on the tree representing a possible decision
2. Decision tree can be used to solve classification problems
3. Decision tree can also be used to solve continuous predictions such as regression

Entropy describes the messiness of the problem being classified. The messier the problem is, the larger the entropy. In decision tree problems, we can use the change in entropy to determine what the decision node is. **The optimum decision node is where the information gain is the largest (i.e. reduces the most entropy)**.

The entropy of decision problem can be calculated as:

$$-\Sigma_{x=1}^{i} p(value_x) log_2(p(value_x))$$

Where value is the proportion of occurrence of one group $value_x = \frac{counts \quad in \quad one \quad group}{total \quad counts}$

Whichever category can reduce the entropy the greatest will be the node for classification.

Data

```
path <- 'https://raw.githubusercontent.com/guru99-edu/R-Programming/master/titanic_data.csv'
titanic <-read.csv(path)

# data cleaning
titanic <- select(titanic, survived, pclass, sex, age)
titanic <- mutate(titanic, survived = factor(survived), age = as.numeric(age))
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

```
# data splitting
set.seed(123)
sample = sample.split(titanic$survived, SplitRatio = 0.7)
train <- titanic[sample, ]
test <- titanic[!sample, ]
```

Modeling

```
tree <- rpart(survived ~., data = train)
```

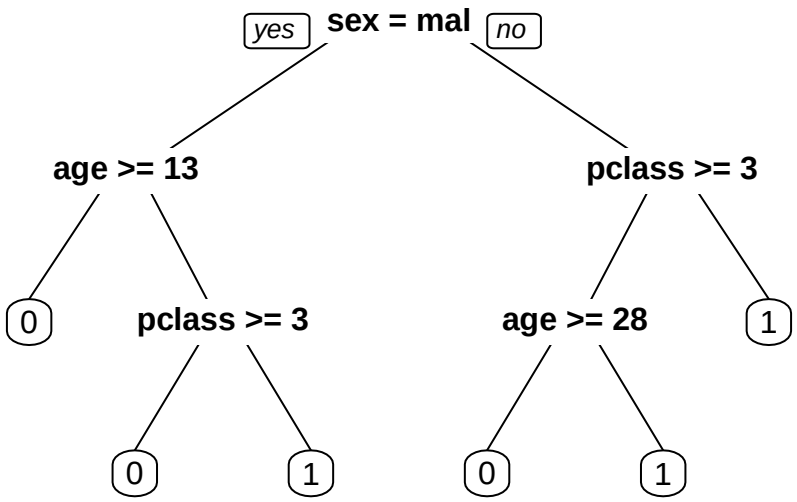
Prediction

```
tree.survived.pred <- predict(tree, test, type = "class")

# evaluate model accuracy
confusionMatrix(tree.survived.pred, test$survived)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 218  47
##          1  25 103
##
##                Accuracy : 0.8168
##                  95% CI : (0.7749, 0.8538)
##     No Information Rate : 0.6183
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.6006
##
##  Mcnemar's Test P-Value : 0.01333
##
##             Sensitivity : 0.8971
##             Specificity : 0.6867
##          Pos Pred Value : 0.8226
##          Neg Pred Value : 0.8047
##              Prevalence : 0.6183
##          Detection Rate : 0.5547
##    Detection Prevalence : 0.6743
##       Balanced Accuracy : 0.7919
##
##        'Positive' Class : 0
##
```

```
# visualize decision tree
prp(tree)
```


yes
sex = mal
no
age >= 13
pclass >= 3
0
pclass >= 3
age >= 28
1
0
1
0
1

Random Forest

Random forest works by building multiple decision trees, which can be applied for classification and regression purposes. Random forest is an ensemble of decision trees that are classified based on different standards, and then which ever classification appears the most in the result (i.e. highest frequency), will be defined as the final decision result.

helpful link: https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/

Data

```
winequality <- read.csv("http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequali

# data manipulation
winequality <- mutate(winequality, quality = as.factor(quality))

# data split
sample <- sample.split(winequality$quality, SplitRatio = 0.7)
train <- winequality[sample, ]
test <- winequality[!sample, ]
```

Modeling

Arguments in the randomForest function:

- mtry: the number of variables randomly sampled as candidate for each decision tree. the recommended mtry value is $mtry = \lfloor \sqrt{ncol(x)} \rfloor$
- ntree: in order to be a tie breaker, we shall set the ntree argument to an odd number. the argument specifies the number of decision trees that will be constructed
- importance: TRUE or FALSE argument, defines whether or not to check the correlations between variables

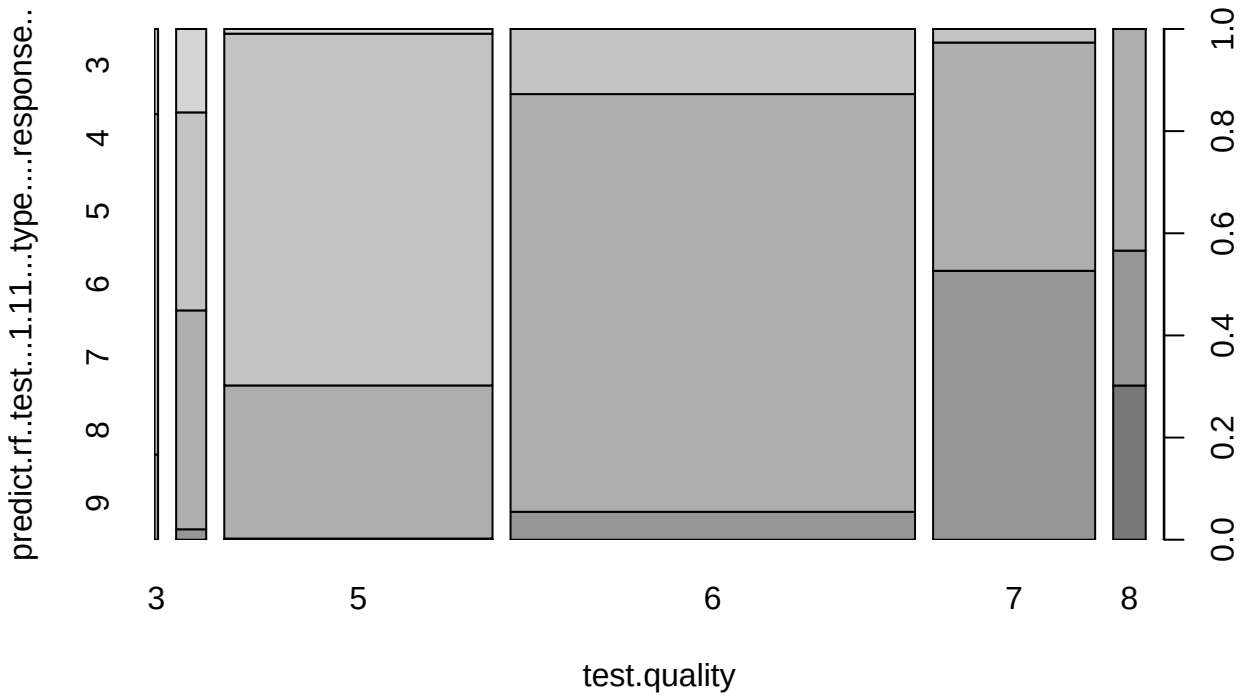
```
rf <- randomForest(quality ~., data = train,
                   mtry = floor(sqrt(ncol(winequality))),
                   ntree = 2001, importance = TRUE)
rf
```

```
##
## Call:
##  randomForest(formula = quality ~ ., data = train, mtry = floor(sqrt(ncol(winequality))),      ntree
##                Type of random forest: classification
##                      Number of trees: 2001
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 32.69%
## Confusion matrix:
##   3  4   5    6   7  8 9 class.error
## 3 0  0   6    8   0  0 0   1.0000000
## 4 0 23  58   33   0  0 0   0.7982456
```

```
## 5 0  4 691  316   9  0 0   0.3225490
## 6 0  1 226 1230  80  2 0   0.2007797
## 7 0  0  11  281 320  4 0   0.4805195
## 8 0  0   1   40  37 44 0   0.6393443
## 9 0  0   0    1   3  0 0   1.0000000
```

Prediction

```
result <- data.frame(test$quality, predict(rf, test[, 1:11], type = "response"))
plot(result)
```




```
# accuracy
sum(result[, 1] == result[, 2])/nrow(result)
```

```
## [1] 0.6827774
```

Support Vector Machine

Support vector machine is a binary classification technique. SVM may be a better classification method than logistic regression when the number of variables is large. SVM is achieved by finding the best separating boundary line by computing the maximum distance margin ($D^- + D^+$) from equidistant support vectors

(i.e. finding points from both groups that are close to each other as support vectors to support the algorithm). The resulting dividing line is called a hyperplane

SVM kernel function will be used when the current dimension does not support linearly separable data points. The kernel function will increase the dimension to where there is possible linear separation

Data

```
data(PimaIndiansDiabetes)
df <- PimaIndiansDiabetes
# for the sake of demonstration, we will use 2 variables (i.e. 2D space)
df <- df[, c("glucose", "insulin", "diabetes")]

# splitting data
split <- caTools::sample.split(df$diabetes, SplitRatio = 0.7)
train <- df[split, ]
test <- df[!split, ]
head(train)
```

```
##   glucose insulin diabetes
## 1     148       0      pos
## 3     183       0      pos
## 4      89      94      neg
## 5     137     168      pos
## 7      78      88      pos
## 9     197     543      pos
```

```
# data pre-processing
train$diabetes <- as.character(train$diabetes)
train$diabetes <- ifelse(train$diabetes == "pos", +1, -1)
head(train)
```

```
##   glucose insulin diabetes
## 1     148       0        1
## 3     183       0        1
## 4      89      94       -1
## 5     137     168        1
## 7      78      88        1
## 9     197     543        1
```

Modeling

```
# finding best linear boundary through SVM
svm.model <- svm(diabetes ~., data = train,
                 type = "C-classification", kernel = "linear",
                 scale = FALSE)
summary(svm.model)
```

```
##
## Call:
```

```
## svm(formula = diabetes ~ ., data = train, type = "C-classification",
##     kernel = "linear", scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  312
##
##  ( 156 156 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Prediction

```
pred <- predict(svm.model, test)
test$diabetes <- ifelse(test$diabetes == "pos", +1, -1)
table(Actual_value = test$diabetes, Predicted_value = pred)
```

```
##              Predicted_value
## Actual_value  -1   1
##           -1 132  18
##           1   42  38
```

Clustering

Hierarchical clustering stops when the grouping result in just one group. Agglomerative clustering is a botton-up approach (Agglomerative clustering begins with each element as a single cluster and gradually grouping them into larger clusters), and Divisive approach is a top-down approach (take an entity of elements in one whole cluster and randomly divide them into two clusters, then calculate cluster sum of squares using $B_{j12} = n_1(\bar{x}_{j1} - \bar{x}_j)^2 + n_2(\bar{x}_{j2} - \bar{x}_j)^2$ where B_j is the distance between the two clusters, $x_{j1}\&x_{j2}$ are the mean of the clusters, n is the number of elements in each cluster, x_j is the grand mean. Which ever separation creates the largest sum of squares will be kept, then repeat the process entire reach desired number of clusters). Hierarchical clustering works through finding the closest (pair with the least distance separation) pair of points and grouping them together (forming the dendogram).

There are different ways of determining the distance between points:

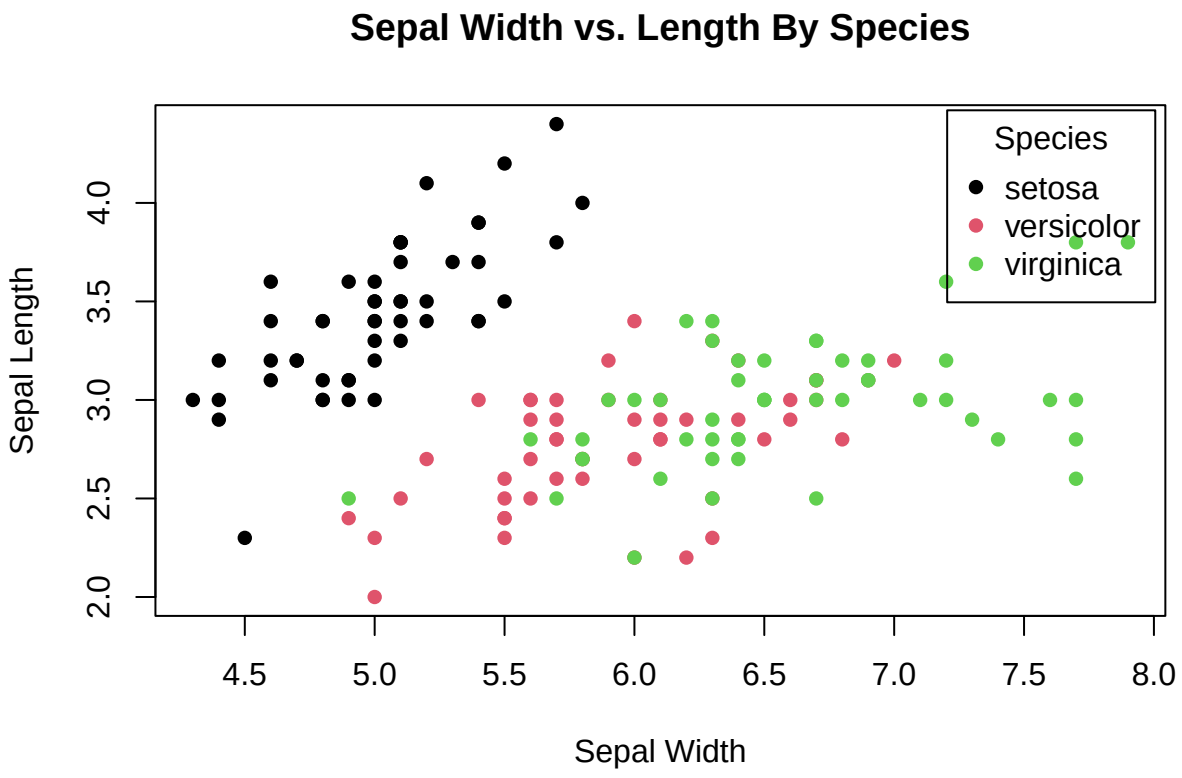
1. Euclidean distance measure: $d = \sqrt{\Sigma_{i=1}^{n}(q_i - p_i)^2}$
2. Squared euclidean distance measure: $d = \Sigma_{i=1}^{n}(q_i - p_i)^2$
3. Manhattan distance measure (sum of the horizontal and vertical components between the two points): $d = \Sigma_{i=1}^{n}|q_x - p_x| + |q_y - p_y|$
4. Cosine distance measure (measures the angle between the two vectors): $d = \frac{\Sigma_{i=0}^{n-1} q_i - p_i}{\Sigma_{i=0}^{n-1}(q_i)^2 \times \Sigma_{i=0}^{n-1}(p_i)^2}$

Key questions for forming clusters:

1. How to represent clusters with more than one point: we use centroids (average of the points)
2. When to stop combining clusters:
   (a) Approach1: predefining the number of clusters needed (K clusters), so the clustering will stop entire K clusters are found. However, the limitation to this method is we need to have prior knowledge about the data so we can predefine the number of clusters we want
   (b) Approach2: stop when the next merge create a cluster with low cohesion
   (c) Approach3: diameter of a cluster is specified, so the clustering will stop when the radius of a new cluster exceeds the predefined threshold

Data

```
data("iris")
plot(iris$Sepal.Length, iris$Sepal.Width, col = iris$Species,
     xlab = "Sepal Width", ylab = "Sepal Length",
     main = "Sepal Width vs. Length By Species", pch = 16)
legend("topright", inset = 0.01,
       legend = c("setosa", "versicolor", "virginica"),
       col = 1:3, title = "Species", pch = 16)
```



Modeling

```r
# scaling
scaled_iris <- scale(iris[, -5])

# calculate Euclidean distance
distance <- dist(scaled_iris)

# dendrogram
dendrogram <- hclust(distance, method = "average")
plot(dendrogram, labels = iris$Species, hang = -1)
```