UNIVERSITÀ
DEGLI STUDI
DI MILANO

STATISTICAL LEARNING

An Overview of Supervised and Unsupervised Analysis

Besim M. Sakaoglu, MAT 985861

September 2022

SUMMARY

The aim of the following analysis is to use Supervised Unsupervised learning and Unsupervised Learning and the associated models with it in order to come up clear statistics from two different data sets. The project will apply and investigate Supervised techniques such as Ridge Regression(L2), LASSO Regression(L1), KNN and Random Forest. Accordingly, we will take a look at Unsupervised Techniques such as K-Mean and Hierarchical Clustering on the same data set respectively.

Supervised Learning

An algorithm known as supervised learning learns from labeled training data to assist in making predictions about unanticipated data. In supervised learning, you teach the computer using appropriately "labeled" data. It indicates that some material has previously been appropriately tagged.

DATA SET-Boston Housing Prices

The U.S. Census Service gathered data for this dataset about housing in the Boston, Massachusetts, area. It was retrieved from the StatLib archive, which can be found at http://lib.stat.cmu.edu/datasets/boston, and has been widely used to compare algorithms in the literature. However, these comparisons were primarily done outside of Delve and are thus somewhat suspect. The dataset is small in size with only 506 cases.

The name for this dataset is simply ‘boston’. It has two prototasks: nox, in which the nitrous oxide level is to be predicted; and price, in which the median value of a home is to be predicted

There are 14 attributes in each case of the dataset. They are:

1. CRIM - per capita crime rate by town
2. ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS - proportion of non-retail business acres per town.
4. CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
5. NOX - nitric oxides concentration (parts per 10 million)
6. RM - average number of rooms per dwelling
7. AGE - proportion of owner-occupied units built prior to 1940
8. DIS - weighted distances to five Boston employment centres
9. RAD - index of accessibility to radial highways
10. TAX - full-value property-tax rate per $10,000
11. PTRATIO - pupil-teacher ratio by town
12. B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT - % lower status of the population
14. MEDV - Median value of owner-occupied homes in $1000's

Ridge regression (L2)

A form of penalized regression (Shrinkage/Regularization methods). Models a linear relationship and assigns penalties for correlated or meaningless coefficients. Linear Regression performs poorly when you have large amount of multivariate data where number of variables greater than number of samples. Ridge regression essentially is Linear Regression but with added constraints to the coefficients. For instance, it will add coefficient value that is relatively close to the zero. As a result, the seemingly meaningless values will approach to zero and we will have less variables to worth with.

Bias-Variance Tradeoff

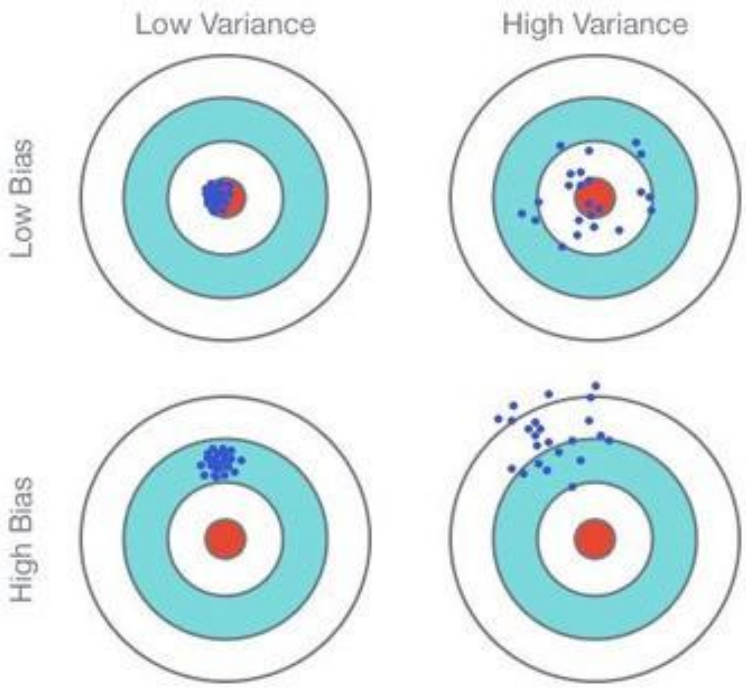
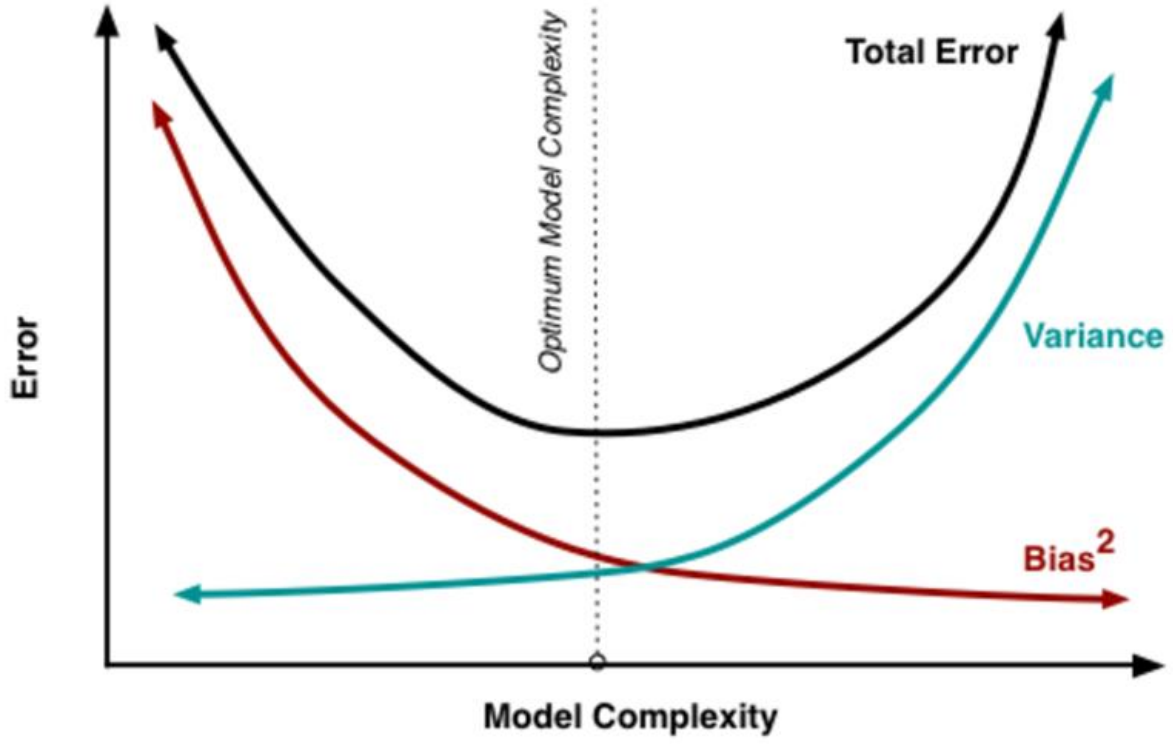


Fig. 1: Graphical Illustration of bias-variance trade-off , Source: Scott Fortmann-Roe., Understanding Bias-Variance Trade-off

Penalization Visualization



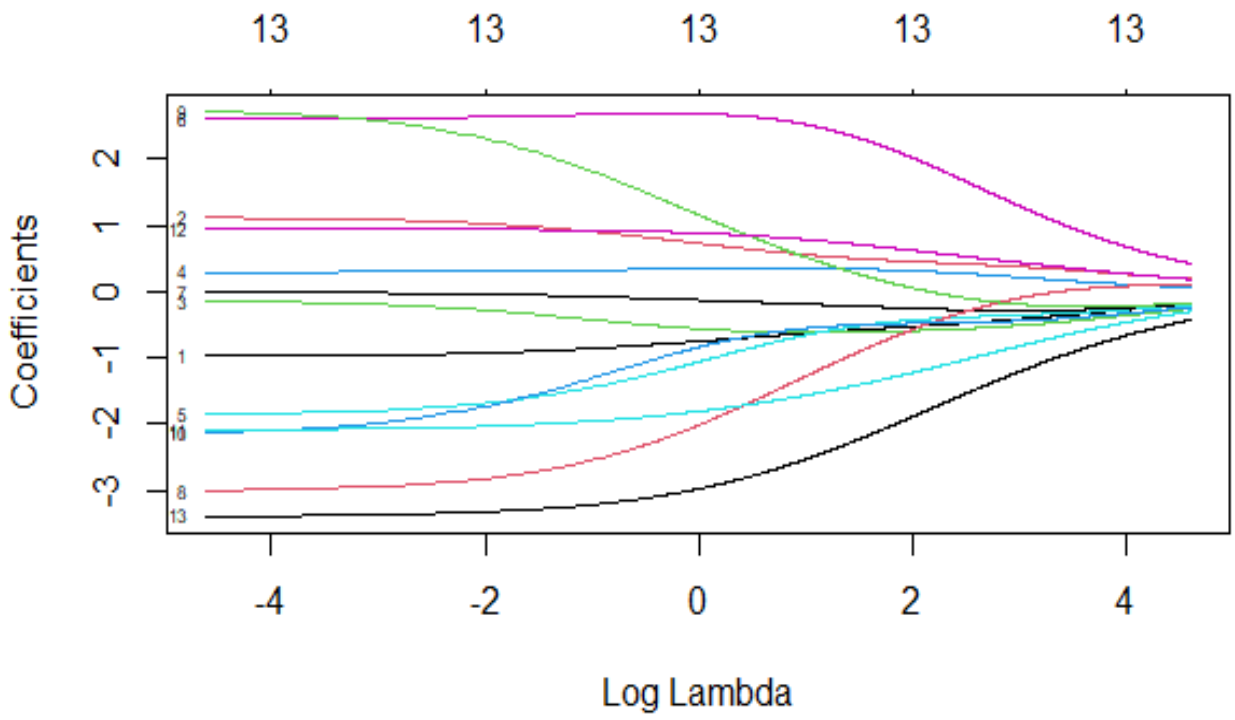
Basically, ‘Model Complexity’ denotes that the number of predictors(X). As we may observe, bias and variance are inversely correlated. If we have high variance, we have lower bias and vice versa. When it comes to finding an optimal point, there is a sweet spot for minimization of the error. We are basically searching for low model complexity and at the same time lowest error.

On the next step, we will scale our data. Scaling, with default settings, will calculate the mean and standard deviation of the entire vector, then "scale" each element by those values by subtracting the mean and dividing by the standard deviation.

After we cleaned our data set, all the data and everything is scaled with its respected proportion. Now we need to separate our data set into Training data and Test data. I picked 80% / 20% split ratio. Meaningly, 404 of the observations for the train indices and remaining part (not chosen from train) is belong to the test indices. Everything is in these indices are randomized picked values, another saying is no duplicate.

Lambda as a tuning parameter

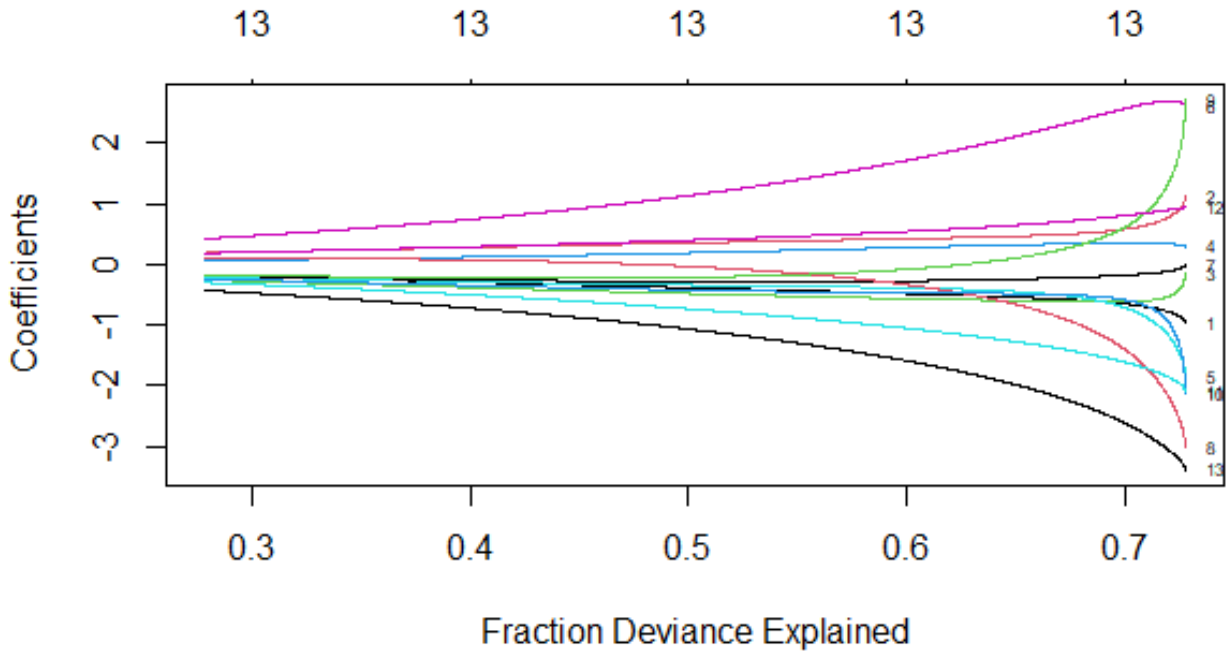
The regularization parameter, known as lambda, indicates the level of regularization. No regularization is produced when lambda is set to 0, but more regularization is produced when lambda is set to big values. We set 0.01 as a starting point for lambda and incrementing by 0.01 up to 100. For finding optimal lambda, in general, we can use cross-validations in relation to robustness of our model. As lambda becomes larger, this will start decreasing the significance of the coefficients (penalty).



What we see on the plot is 13 different colours represent all of the variables that currently inside of the model in terms of relevancy. So, each stage of Log Lambda we have all of our models that are included all of our features in terms of like one all the way to 13 somewhere they all are here, but they all will go slowly shrinking to zero as the more Lambda we have. So this would be like a significant increase for that specific area in terms of whether or not that specific variable is relevant to the model. In this case, since numbers 6 and 13 are not shrinking as much as the others, we can easily say that those are more relevant to our model compared to the others. Additionally, one of our least significant values like coefficient number 3 (INDUS) which goes toward to zero much more quickly then the other variables involved in our overall model. Basically, when Lambda (Log Lambda) increases more and more to the right then all of our coefficients are going to go towards zero and diminishes.

Goodness of Fit

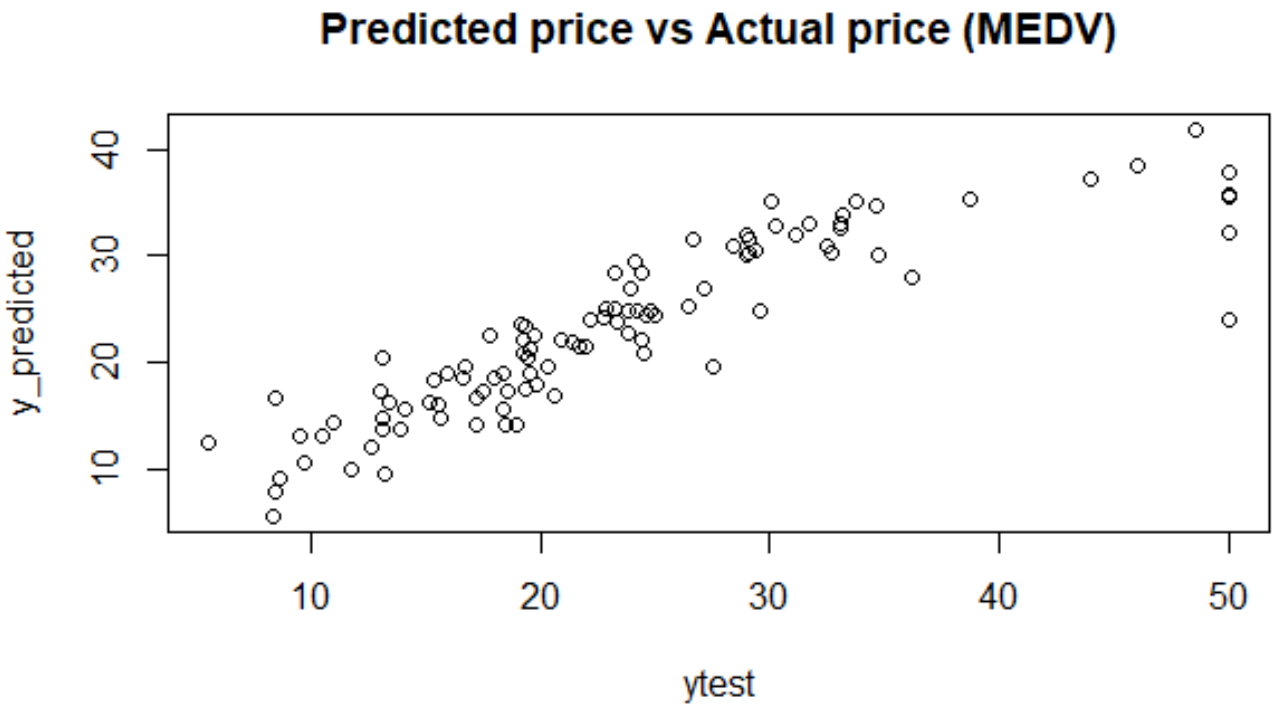
Let`s take a look at the goodness of fit:



We simply can say that there is a significant variance that straying away from the given. In this case, coefficients start to diverge further and further out.

We can investigate coefficient determination by calculating $\boldsymbol{R^2}$. We can assign Total Sum of Squares by constructing the formula on the compiler(Rstudio) and same is valid for the Sum of Squares Error. When we have done this, we ended up with as a result **0.7615991** So, our model explains almost **76%** of the variance. Accordingly, We easily can check the Mean Square Error(MSE) which is around 24 for this particular case.

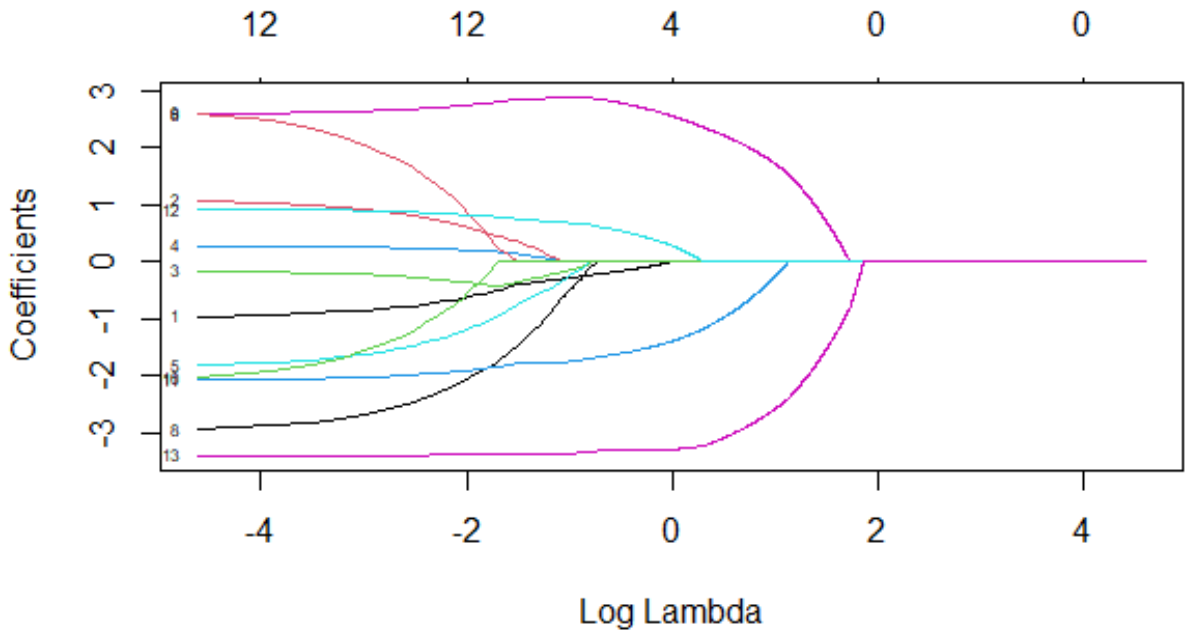
Ultimately we can take a look at the Predicted and Actual prices :



There is a good match with predicted and actual prices though there are some outliers as we may observe on the figure above.

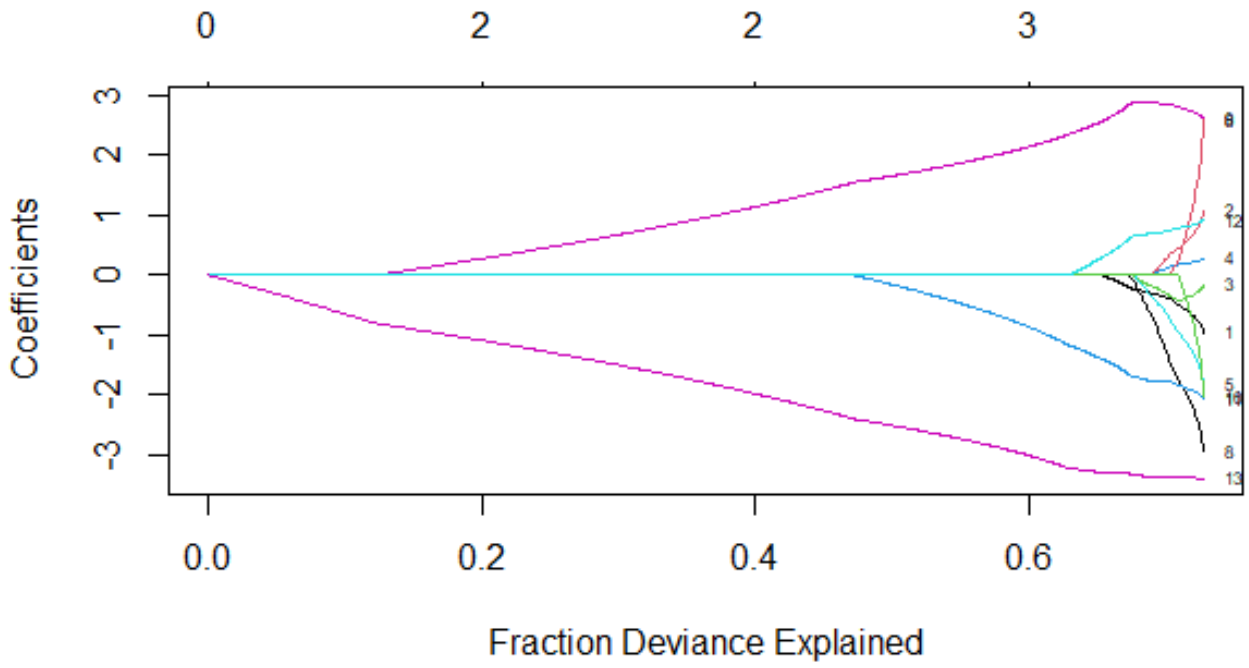
LASSO (L1) Theory

It is a form of penalized regression. However, penalizes high coefficients unlike Ridge Regression. It forces coefficients to be 0 whereas Ridge Regression approaches to zero. Moreover, it avoids multi-collinearity, picks one variable and shrinks others



Since this is a LASSO regression all the values that appear to touch zero and they remain zero for the duration of our lambdas as they increase unlike Ridge regression. The same theory applies to the situation on coefficient number '13' and on the coefficient number '6' but they dies slower compared to the others do. As we mentioned before, it denotes that those features are crucial to our overall model.

Goodness of Fit-LASSO



Essentially what we have here is that as the coefficients are increasing and then they are slowly decreasing because they are penalized due to they are in the LASSO. At The point 0.6, 60% of the variation explained by those particular coefficients that are associated with each of these features and if we have all of the features involved then we are going to have close to one(100%) in terms of explanation but in terms of robustness if there is new data coming in then we will have relying on benchmark such as coefficent number 13, coefficient number 6 and if needed, coefficient number 10(blue one) we are going to using those in terms of identifying the overall variance if we want to remove other features that are not as relevant.

Conclusion

Likewise we did in the Ridge part, we will investigate coefficient determination by calculating $\boldsymbol{R^2}$. We will assign Total Sum of Squares by constructing the formula on the compiler(Rstudio) and same is valid for the Sum of Squares Error. When we have done this, we ended up with as a result **0.7611498** So, our model explains, again, around **76%** of the variance. When we check the Mean Square Error(MSE), we can say that Ridge is a little bit better than compared to LASSO in terms of Mean Squared Error since it has lower value of MSE. Since they generates almost exactly identical varience explenation and there is not much of a difference between the prediction output for both LASSO and Ridge regressions we can use both of them on this data set.

Data Set Description

Iris Plants Database

The Iris flower data set or Fisher's Iris data set is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper. This is perhaps the best-known data set at the Statistics or Data Science jargon. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

Number of Instances: 150 (50 in each of three classes)

Number of Attributes: 4 numeric, predictive attributes and the class

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:

-- Iris **Setosa**

-- Iris **Versicolour**

-- Iris **Virginica**

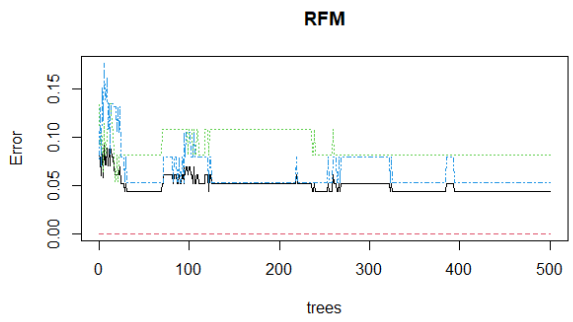
Missing Attribute Values: None

Summary Statistics:

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490
petal width:	0.1	2.5	1.20	0.76	0.9565

Random Forest- Supervised Learning

We will apply Random Forest technique to the Iris data set. We will split our data in this way: 70% of the data is assigned to the training set and 30% of the data is assigned to the training set. When we done with the process, we can investigate the model accuracy. Initially, we can check Confusion Matrix below.



	setosa	versicolor	virginica
setosa	13	0	0
versicolor	0	8	1
virginica	0	0	17

We need to look at the diagonal since the diagonal elements represent the number of points for which the predicted label is equal to the true label. It basically suggests correct prediction. So, we easily can say that model prediction for '**setosa**' and '**versicolor**' have been correctly classified. However, there is a mismatch for one 'virginica' labeled as versicolor.

Seeking for further investigations, we can look at Classification Accuracy. It is basically sum of the observations on the diagonal values divided by the sum of all observations included in the Confusion Matrix. The accuracy of the Random Forest model built to predict species using input variables such as Sepal Length, Sepal Width, Petal Length and Petal Width is around 97.4% which is pretty neat.

KNN-supervised

Choosing ‘k’

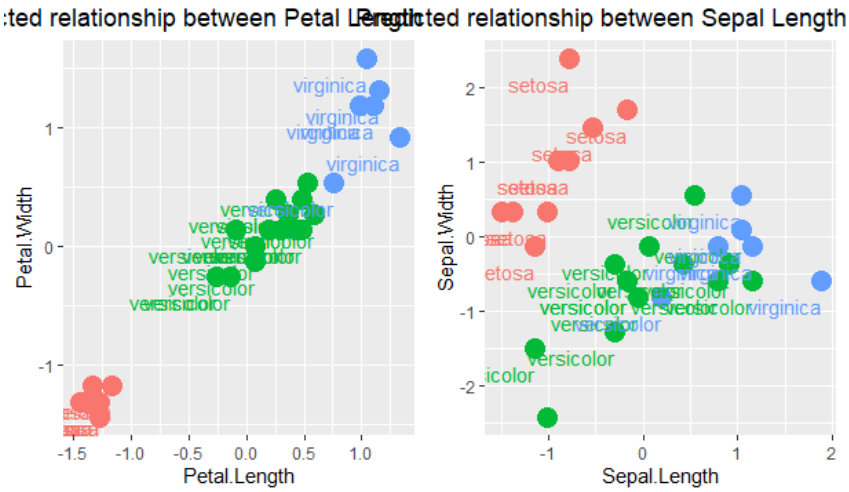
We can choose ‘k’ number of data points where k represents the number of nearest neighbors that are closest to a given observation. When the question comes our mind “How can we find a good starting point for the k value?”

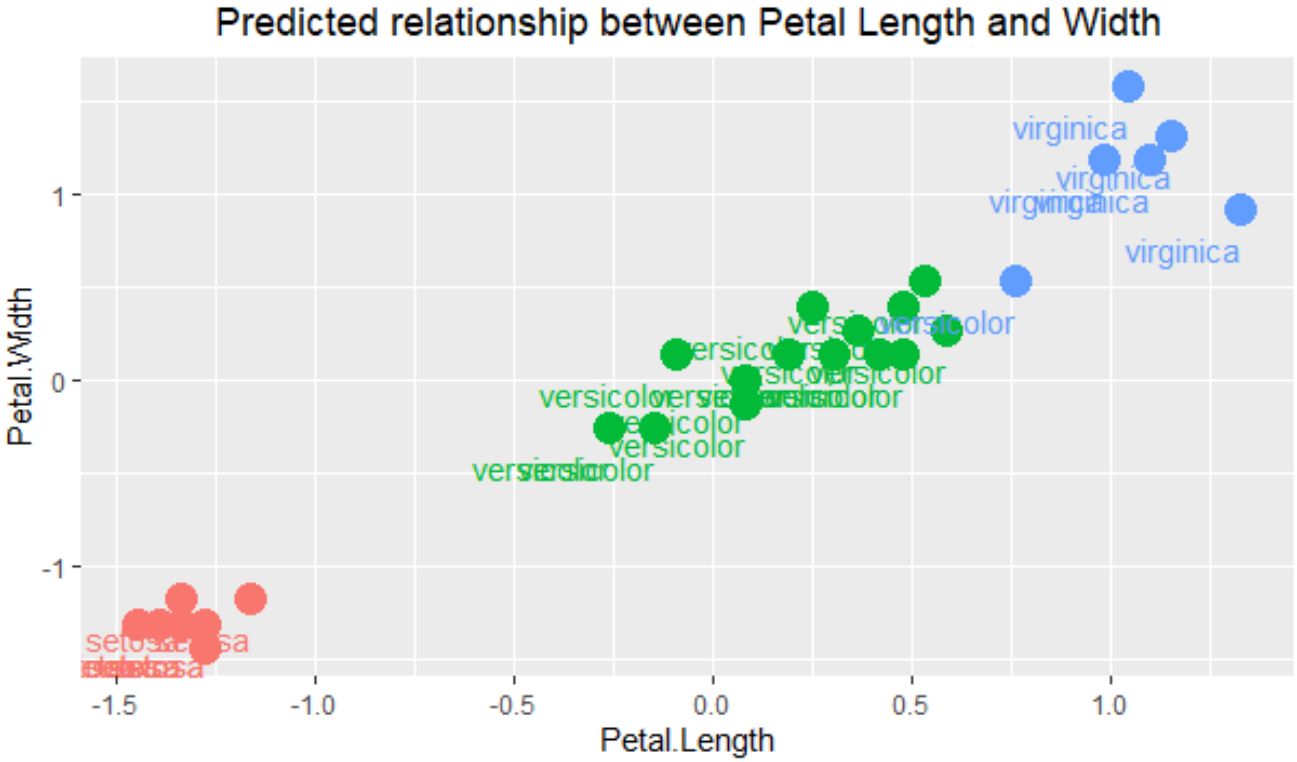
We simply can take the square root of the number of observations that in our data set. In general, K should be odd since the algorithm might confuse for even numbers of classes. Every instance essentially submits a vote for their class, and the prediction belongs to the class that receives the most votes. It is a good idea to select a K value with an odd number if we are using K and we have an even number of classes (for example, 2) in order to prevent a tie. That is, any even number of K can lead to uncertainty if the neighbors are 50% of one and 50% of the second class.

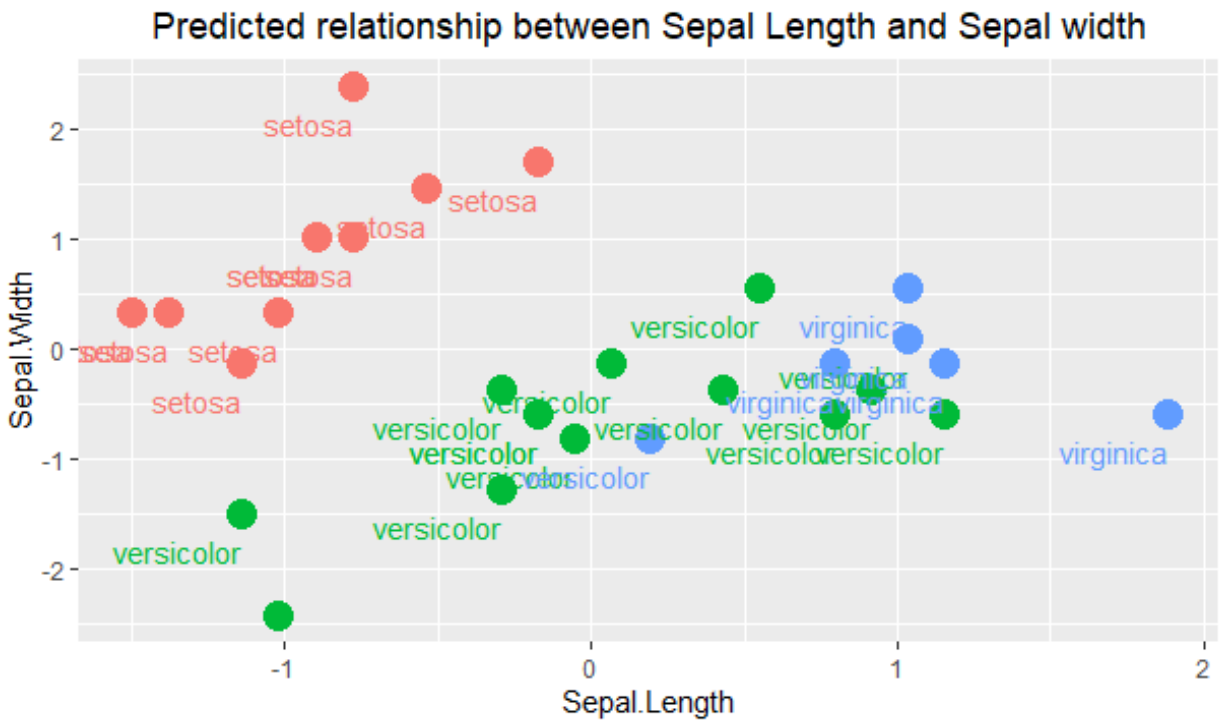
There is no structured method on picking the best value of k but we can always parameterize this term by using a grid search algorithm or where we just have an array and our array consists of k values

We can choose an arbitrary value and just go from there. Additionally, we can choose to do cross-validation on our data to have a more robust model. Some things to note about k, let say that k is equal to one. This will actually more or less underrepresent our data since all new data that will be added to our model will vary very widely and this is unlikely situation indeed.

The other way around is that if we choose very large values of k, it will end up with very computationally expensive, and it also may overfit your data. The reason why is because you are calculating all the relationships inside of your data set.


Predicted relationship between Petal Length
Predicted relationship between Sepal Length
Petal.Width
Petal.Length
Sepal.Width
Sepal.Length
virginica
versicolor
setosa


Predicted relationship between Petal Length and Width
Petal.Width
Petal.Length
virginica
versicolor
setosa



Conclusion

When we look at the plot we can say that our KNN algorithm performed as Random Forest model did. ‘setosa’ and ‘versicolor’ are identified almost perfectly. However, we may see that ,again, one ‘versicolor’ colored blue instead of green but the overall outcome was satisfying enough. Since their accuracy is legitimate both knn and random forest techniques are usable for this data set as a supervised learning method.

Unsupervised Learning

Unsupervised learning is the process by which artificial neural networks attempt to find similarities between given data sets. In unsupervised learning, computers try to do this. Analysis is limited to catching and highlighting clusters and patterns rather than being based on specific predictions of the dependent variable.

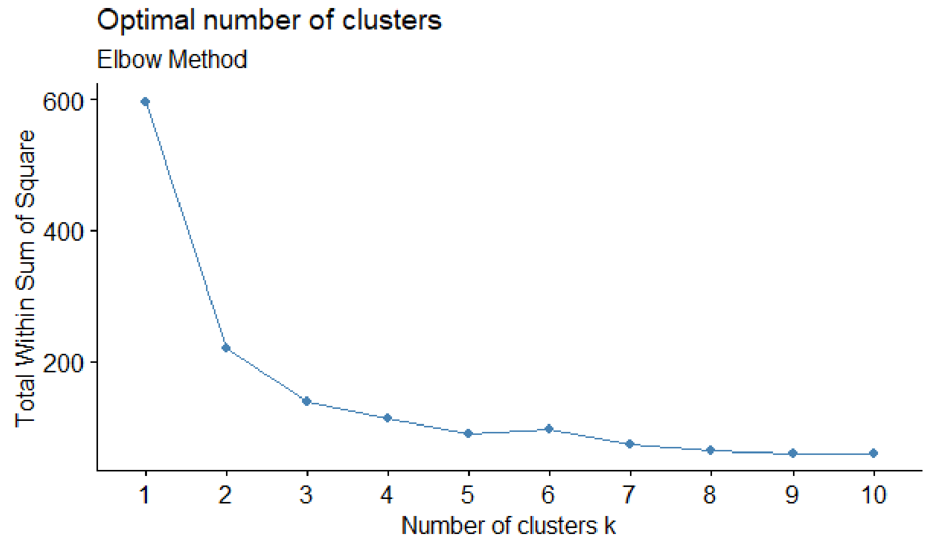
K-means

The K-Means clustering technique is one of the most popular unsupervised learning techniques used to identify clusters of data objects in a dataset. It basically uses a fixed number of clusters (centroid), group together observations based on similarities. Additionally, it uses Euclidean distance. It randomly configures K centroids in the data space, based on these positions, the algorithm will optimize for the best position of the centroids. Accordingly, algorithm stops when no change in centroid values occurred or, the number of iterations has been reached. The hypothesis behind the model for selecting parameters relies on K estimation based on the elbow approach. The empirical approach is performed as follows: compute the sum of the within-cluster squared errors (WSS) for different values of k and choose the k at which the WSS starts decreasing first. In the WSS-vs-k representation, this appears as an elbow.

For applying K-Means technique, we will use the same data set called “Iris Data Set” that introduced in previous parts in this paper. However, we need to arrange our data set in order to apply K-Means since K-Means is an Unsupervised technique, we need to unlabel the data like Setosa, Virginica, Versicolor that included in the last column. (iris_data <- iris[1:4])

SCALING DATA

It is very important because we want the distance matrix relatively unweighted and when we are scaling our data will be more balanced data set since we are going to use Euclidean distance which is default distance used.



Based on this graph we can now utilize our knowledge of the ‘elbow looking’ plots and determine which is a good point to identify the number of clusters ‘k’.

We already know that there are 3 types of clusters (setosa, virgicolar and virginica). So, we can use 3 clusters by choosing k=3 as well as 4 or 5. When we are looking at this elbow plot we are essentially try to determine the however many clusters that we need to use, we need to look at the **diminishing returns**.

Clustering vector:

[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

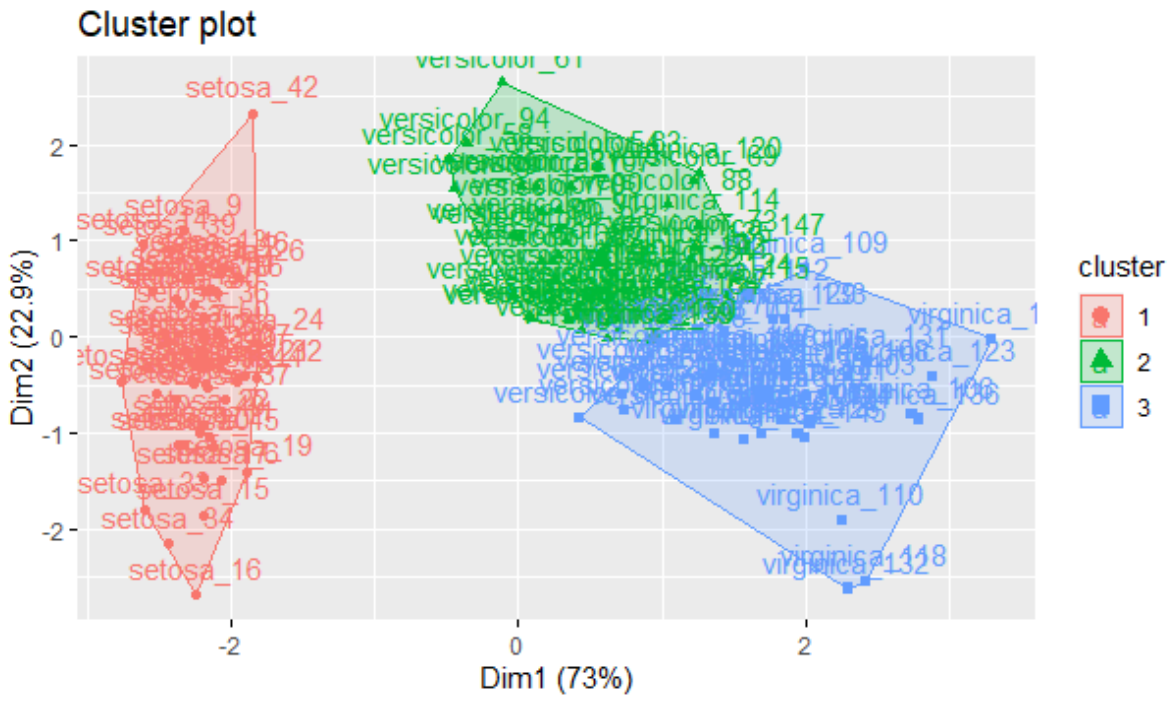
[40] 1 1 1 1 1 1 1 1 1 1 1 3 3 3 2 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2 3 2 2 2 2 3 3 3

[79] 2 2 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 3 3 2 2 3 3

[118] 3 3 2 3 2 3 2 3 3 2 3 3 3 3 3 3 2 2 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2

This provides us the type of group that each of our observations are part of. Also, we easily can assign this to an additional feature within our data set.

We want each of these rows to be unique, we do not want any identical row names in another saying. Simply, we can change the row names related to its species. It will also contribute to better visualisations, indeed.



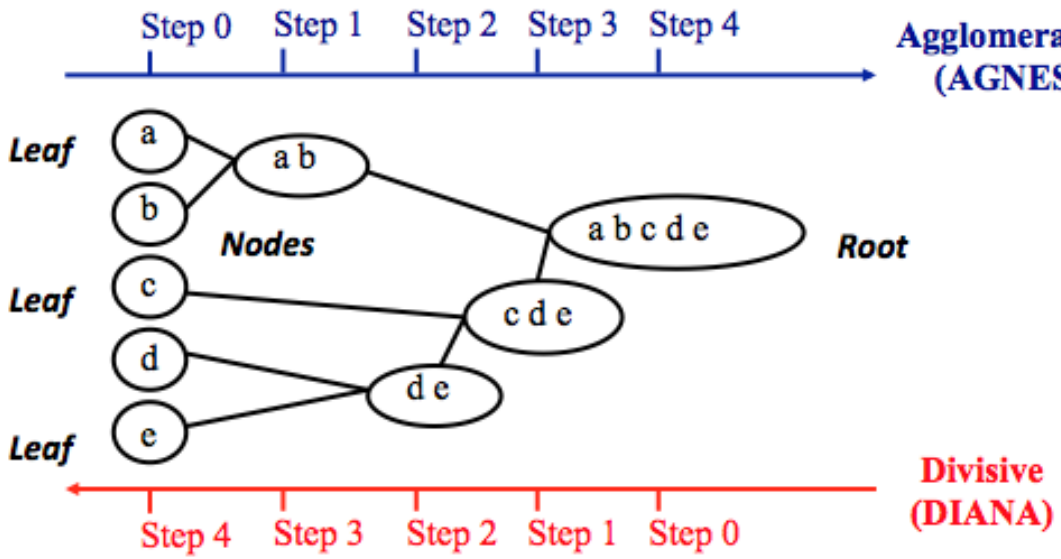
When we look at the Cluster Sum of Squares, we can observe almost 77% classification(identification) success rate within clusters.

We can observe for ‘setosa’, all identified correctly. However, we have some mismatches between ‘virginica’ and ‘versicolor’ as we may see here:

km.clusters	setosa	versicolor	virginica
1	50	0	0
2	0	39	14
3	0	11	36

HIERARCHICAL CLUSTERING

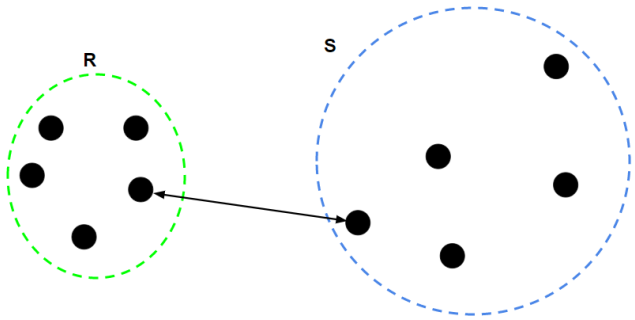
It is also known as hierarchical clustering, is an algorithm that groups similar objects into groups called clusters. An endpoint is a set of clusters, where each cluster is different from every other cluster, and the objects within each cluster are broadly e each other. Moreover, K-means clustering requires us to pre-assign the number of clusters 'K' and it might terminate handicapped outcomes. So, Hierarchical clustering is an alternative approach that does not require pre-assignment 'K'. There are two types of Hierarchical clustering. One of them is Agglomerative Clustering and the other is Divisive Hierarchical Clustering. Agglomerative clustering is the most common type of hierarchical clustering and is used to group objects into clusters based on similarity. Also known as AGNES (Agglomerative Nesting). The algorithm first treats each object as a singleton cluster. Pairs of clusters are then successively merged until all clusters are merged into one large cluster containing all objects. The result is a tree-based representation of the object called a dendrogram. The divisive hierarchical clustering known as DIANA (Divisive Analysis) is the inverse of agglomerative clustering and it starts by including all objects in a single large cluster. At each step of iteration, the most heterogeneous cluster is divided into two segments. Respectively, the process is iterated until all objects are in their own cluster. Practically, divisive clustering is good at identifying large clusters while agglomerative clustering is good at identifying small clusters. There are four Linkage Types.



(Fig. Source: datanovia.com)

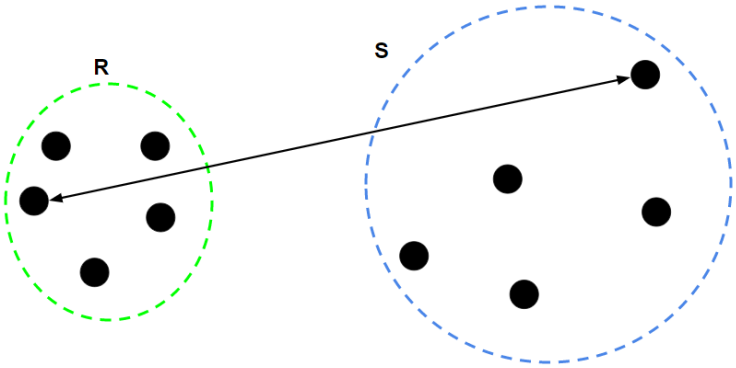
1. Single Linkage: For two clusters R and S, the single linkage returns the minimum distance between two points i and j such that i belongs to R and j belongs to S.

$L(R,S) = \min(D(i,j)), i \in R, j \in S$



2. Complete Linkage: For two clusters R and S, the complete linkage returns the maximum distance between two points i and j such that i belongs to R and j belongs to S.

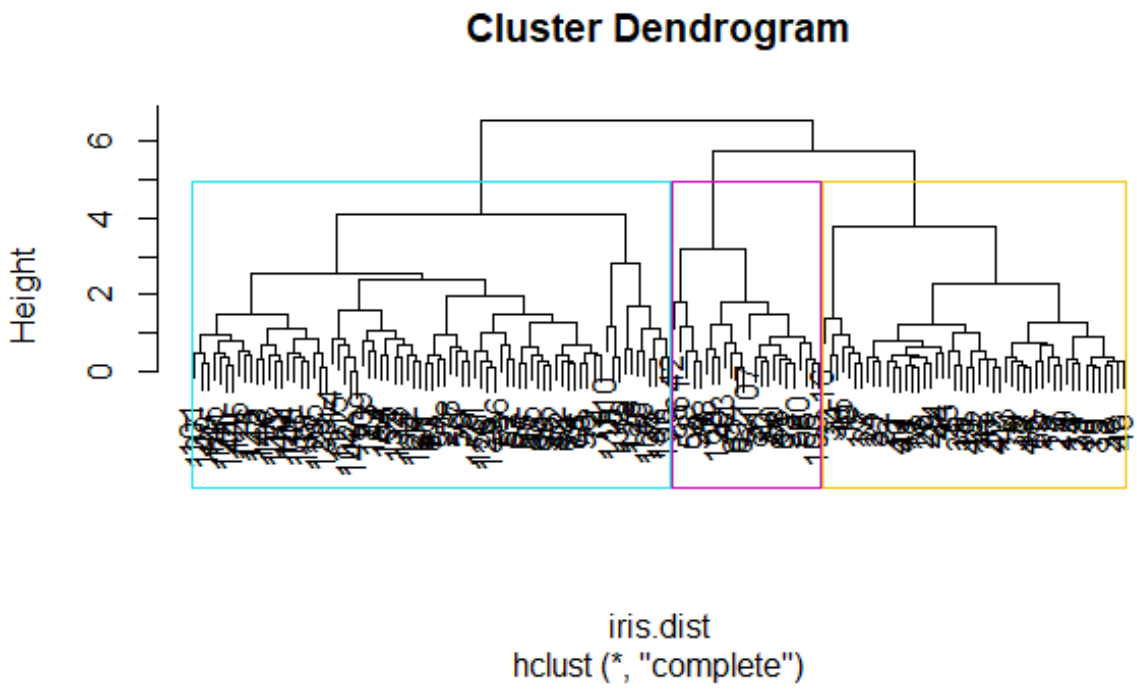
$L(R,S) = \max(D(i,j)), i \in R, j \in S$



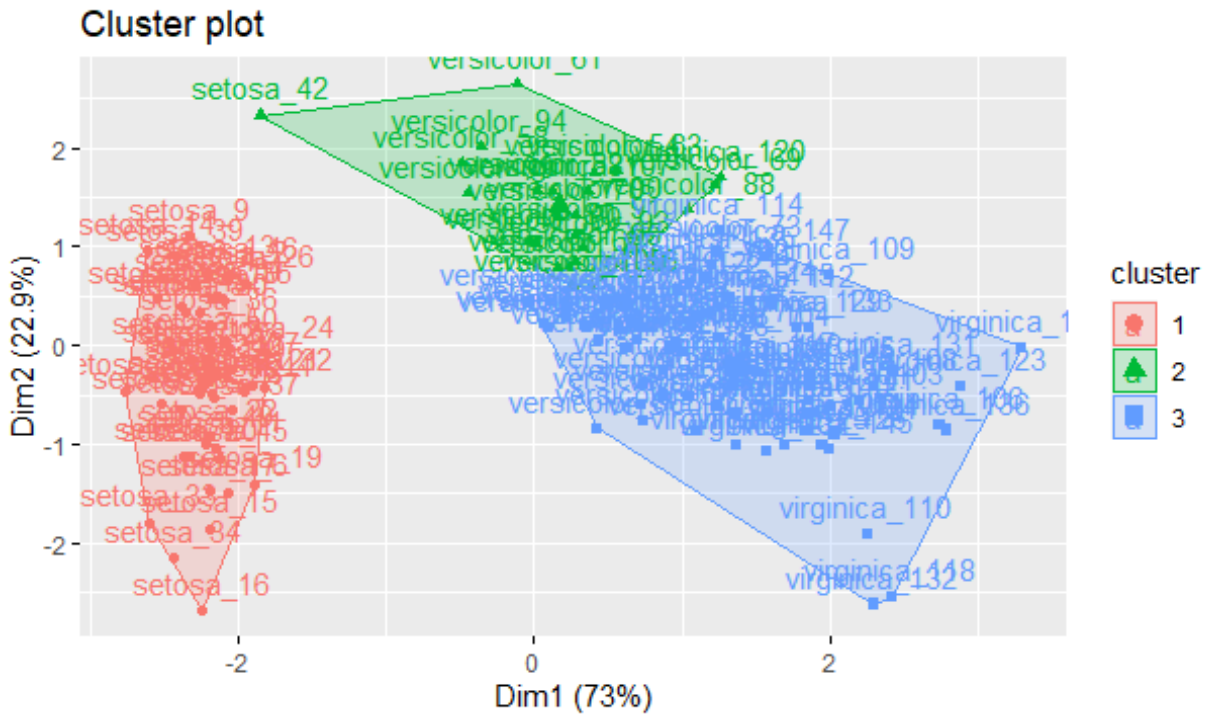
3. Average Linkage: For two clusters R and S, first for the distance between any data-point i in R and any data-point j in S and then the arithmetic mean of these distances are calculated. Average Linkage returns this value of the arithmetic mean.

4. Centroid Linkage : Dissimilarity between the centroid for cluster A (a mean vector of length) and the centroid for cluster B. Centroid linkage can result in undesirable inversions.

- Cluster method : **complete**
- Distance : **euclidean**
- Number of objects: **150**



As we may observe from the 'Cluster Dendrogram Figure' it is a bottom up (agglomerative) approach. Essentially, each of our observations being merged as they go up and up until everything under one group. One of the advantageous things about using the Hierarchical clustering method is we have a really neat tree observation to determine which type or however many types of clusters that we might want to utilize. When we look at the figure above, we easily can observe three different colored clusters since we picked k equals 3.



iris.clusters	setosa	versicolor	virginica
1	49	0	0
2	1	21	2
3	0	29	48

Conclusion

Compared to K-means cluster plot outcome there are some differences. Again, ‘setosa’ identified almost perfectly. However, ‘Versicolor’ could not performed well enough. Basically, we can go with ‘*K-means clustering*’ instead of ‘*Hierarchical clustering*’.

APPENDIX : R Code

Random Forest on iris data set

```
# Install pre-requisite packages

library(stats)
library(dplyr)
library(randomForest)

mydata <- iris
View(mydata)
str(mydata)
#Splitting data in training and testing 80/20 = 97% (for 75/25 = 95.5% )

index = sample(2, nrow(mydata), replace= TRUE, prob=c(0.7,0.3))

#Training data
Training = mydata[index==1,]

#Testing data
Testing = mydata[index==2,]

#Random Forest Model
```

```
RFM= randomForest(Species~. ,  data = Training)

#Evaluating Model Accuracy
Species_Pred = predict(RFM,Testing)
Testing$Species_Predict = Species_Pred
View(Testing)

#Building confusion matrix
CFM=table(Testing$Species,Testing$Species_Predict)
CFM

#Acccuracy

Classification_Accuracy = sum(diag(CFM)/sum(CFM))
Classification_Accuracy
```

K Nearest Neighbors

```
# Importing the dataset
data = iris

row_labels = data[,5]

# Encoding the target feature as factor
data$Species <- as.numeric(data$Species)
```

```
# Scale the data since we will be using distance formulas on the data
# and we want to reduce complexity and computation when computing
# especially when our datasets are huge!
data[,c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width")] <-
scale(data[,c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width")])

# Split into test and train 80/20
set.seed(123)

size <- floor(0.8 * nrow(data))


train_ind <- sample(seq_len(nrow(data)), size = size)

train_labels <- data[train_ind, 5]

data_train <- data[train_ind,1:4]
data_test <- data[-train_ind,1:4]

data_test_labels <- row_labels[-train_ind]
# Fit KNN Model
library(class)

predictions <- knn(train = data_train,
test = data_test,
cl = train_labels,
k= 11)

# Notice that I am only getting 2 dimensions
```

```
plot_predictions <- data.frame(
data_test$Sepal.Length,
data_test$Sepal.Width,
data_test$Petal.Length,
data_test$Petal.Width,
predicted = predictions)

colnames(plot_predictions) <- c("Sepal.Length",
"Sepal.Width",
"Petal.Length",
"Petal.Width",
'predicted')
# Visualize the KNN algorithm results.
library(ggplot2)
library(plyr)
require(gridExtra)

p1 <- ggplot(plot_predictions, aes(Petal.Length, Petal.Width, color = predicted,
fill = predicted)) +
geom_point(size = 5) +
geom_text(aes(label=data_test_labels),hjust=1, vjust=2) +
ggtitle("Predicted relationship between Petal Length and Width") +
theme(plot.title = element_text(hjust = 0.5)) +
theme(legend.position = "none")


p2 <- ggplot(plot_predictions, aes(Sepal.Length, Sepal.Width, color = predicted,
fill = predicted)) +
geom_point(size = 5) +
geom_text(aes(label=data_test_labels),hjust=1, vjust=2) +
ggtitle("Predicted relationship between Sepal Length and Sepal") +
```

```
theme(plot.title = element_text(hjust = 0.5)) +
theme(legend.position = "none")

grid.arrange(p1, p2, ncol=2)
```

Unsupervised – K-means Clustering

```
library(factoextra),

iris
# iris.labels
iris.labels = iris$Species
table(iris.labels)
iris_data <- iris[1:4]

# Scale data
iris_data_scale <- scale(iris_data)
# Distance
iris_data <- dist(iris_data_scale)

# Calculate how many clusters you need
# Within Sum Squares
fviz_nbclust(iris_data_scale, kmeans, method = "wss")+
labs(subtitle="Elbow Method")

# Kmeans
km.out <- kmeans(iris_data_scale, centers=3,nstart=100)
print(km.out)
```

```
# Visualize the clustering algorithm results.
km.clusters<-km.out$cluster
rownames(iris_data_scale)<-paste(iris$Species, 1:dim(iris)[1], sep = "_")
fviz_cluster(list(data=iris_data_scale, cluster = km.clusters))
table(km.clusters, iris$Species)
```

Hierarchical clustering

```
library(factoextra)

iris.labels <- iris$Species
table(iris.labels)

#Data
iris_data = iris[1:4]

#Scale
iris_data_std = scale(iris_data)

#distance

iris.dist = dist(iris_data_std)

#Hierarchical clustering algorithm
```

```
hc.out_iris <- hclust(iris.dist, method = "complete")
hc.out_iris

#Dendrogram
plot(hc.out_iris)
rect.hclust(hc.out_iris, k = 3, border = 5:28)

#Clusters
iris.clusters <- cutree(hc.out_iris, k=3)

#Visualize the cluster
rownames(iris_data_std) <- paste(iris$Species, 1:dim(iris)[1], sep = "_")

fviz_cluster(list(data = iris_data_std , cluster = iris.clusters))


table(iris.clusters, iris$Species)
```

Ridge-supervised

```
# Clear the workspace
rm(list = ls())

# Ridge Regression Example
#setwd('')
# Link of where data came from
#http://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html
```

```
# Setwd with dataset

library(modelr)

data <-
read.csv("https://raw.githubusercontent.com/Scavetta/conf_tensorflow_training_day
1/master/1_Deep_Learning_Intro/data/boston_keras.csv")

str(data)

summary(data)


data <- na.omit(data)



data_scaled <- cbind(scale(data[,1:13]),data[,14])


# Train/set 80/20

set.seed(123)


size <- floor(0.8 *  nrow(data_scaled))


train_ind <- sample(seq_len(nrow(data_scaled)), size = size)


train <- data_scaled[train_ind, ]

xtrain <- train[,1:13]

ytrain <- train[,14]


# Create the values that were 'not' chosen

# Test values

test <- data_scaled[-train_ind,]

xtest <- test[,1:13]

ytest <- test[,14]
```

```
lambda.array <- seq(from = 0.01, to = 100, by = 0.01)



library(glmnet)
ridgeFit <- glmnet(xtrain,ytrain, alpha = 0, lambda = lambda.array)
summary(ridgeFit)

# As lambda becomes larger, this will start decreasing the sign. of the
coefficients
plot(ridgeFit, xvar = 'lambda', label = T)



# Goodness of fit
plot(ridgeFit, xvar = 'dev', label = T)

# Predicted Values
y_predicted <- predict(ridgeFit, s = min(lambda.array), newx = xtest)
# Coefficients
predict(ridgeFit, s = min(lambda.array), newx = xtest, type = 'coefficients')



# SST SSE
sst <- sum((ytest - mean(ytest))^2)
sse <- sum((y_predicted - ytest)^2)

rsquare <- 1 - (sse/sst)



# MSE
```

```
MSE = (sum((y_predicted - ytest)^2) / length(y_predicted))
MSE


plot(ytest, y_predicted, main = 'Predicted price vs Actual price (MEDV)')
```

Lasso-Supervised

```
data <-
read.csv("https://raw.githubusercontent.com/Scavetta/conf_tensorflow_training_day
1/master/1_Deep_Learning_Intro/data/boston_keras.csv")
str(data),
summary(data)


data <- na.omit(data)



data_scaled <- cbind(scale(data[,1:13]),data[,14])


# Train/set 80/20
set.seed(123)


size <- floor(0.8 * nrow(data_scaled))


train_ind <- sample(seq_len(nrow(data_scaled)), size = size)


train <- data_scaled[train_ind, ]
xtrain <- train[,1:13]
```

```
ytrain <- train[,14]

# Create the values that were 'not' chosen
# Test values
test <- data_scaled[-train_ind,]
xtest <- test[,1:13]
ytest <- test[,14]

lambda.array <- seq(from = 0.01, to = 100, by = 0.01)



library(glmnet) # Elastic-Net Regularized Generalized Linear Models
ridgeFit <- glmnet(xtrain,ytrain, alpha = 0, lambda = lambda.array)
summary(ridgeFit)

# As lambda becomes larger, this will start decreasing the sign. of the
coefficients
plot(ridgeFit, xvar = 'lambda', label = T)



# Goodness of fit
plot(ridgeFit, xvar = 'dev', label = T)

# Predicted Values
y_predicted <- predict(ridgeFit, s = min(lambda.array), newx = xtest)
# Coefficients
predict(ridgeFit, s = min(lambda.array), newx = xtest, type = 'coefficients')



# SST SSE
```

```
sst <- sum((ytest - mean(ytest))^2)
sse <- sum((y_predicted - ytest)^2)

rsquare <- 1 - (sse/sst)


# MSE
MSE_ridge = (sum((y_predicted - ytest)^2) / length(y_predicted))
MSE_ridge

rmse_ridge <- sqrt(MSE_ridge)

plot(ytest, y_predicted, main = 'Predicted price vs Actual price (MEDV)')


library(glmnet)

lassoFit <- glmnet(xtrain,ytrain, alpha=1, lambda=lambda.array)
summary(lassoFit)

# Lambdas in relation to the coefficients
plot(lassoFit, xvar = 'lambda', label=T)

# Goodness of fit
plot(lassoFit, xvar = 'dev', label = T)

# Predicted Values
y_predicted_lasso <- predict(lassoFit, s=min(lambda.array), newx = xtest)

# SSE, SST
```

```
sst <- sum((ytest - mean(ytest))^2)
sse <- sum((y_predicted - ytest)^2)

rsquare <- 1 - (sse/sst)


# MSE
MSE = (sum((y_predicted - ytest)^2) / length(y_predicted))
MSE


plot(ytest, y_predicted, main = 'Predicted price vs Actual price (MEDV)')
```

########LASSO REGRESSION Code

```
# Clear the workspace
rm(list = ls())


#setwd('')
# Link of where data came from
#http://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html

# Setwd with dataset
library(modelr)
```

```
data =
read.csv("https://raw.githubusercontent.com/Scavetta/conf_tensorflow_training
_day1/master/1_Deep_Learning_Intro/data/boston_keras.csv")
summary(data)


data_scaled <- cbind(scale(data[,1:13]),data[,14])


# Train/set 80/20
set.seed(123)


size <- floor(0.8 *  nrow(data_scaled))


train_ind <- sample(seq_len(nrow(data_scaled)), size = size)


train <- data_scaled[train_ind, ]
xtrain <- train[,1:13]
ytrain <- train[,14]


# Create the values that were 'not' chosen
# Test values
test <- data_scaled[-train_ind,]
xtest <- test[,1:13]
ytest <- test[,14]


lambda.array <- seq(from = 0.01, to = 100, by = 0.01)



library(glmnet)
ridgeFit <- glmnet(xtrain,ytrain, alpha = 0, lambda = lambda.array)
summary(ridgeFit)
```

```
# As lambda becomes larger, this will start decreasing the sign. of the
coefficients
plot(ridgeFit, xvar = 'lambda', label = T)



# Goodness of fit
plot(ridgeFit, xvar = 'dev', label = T)

# Predicted Values
y_predicted <- predict(ridgeFit, s = min(lambda.array), newx = xtest)
# Coefficients
predict(ridgeFit, s = min(lambda.array), newx = xtest, type = 'coefficients')



# SST SSE
sst <- sum((ytest - mean(ytest))^2)
sse <- sum((y_predicted - ytest)^2)

rsquare <- 1 - (sse/sst)



# MSE
MSE_ridge = (sum((y_predicted - ytest)^2) / length(y_predicted))
MSE_ridge

rmse_ridge <- sqrt(MSE_ridge)

plot(ytest, y_predicted, main = 'Predicted price vs Actual price (MEDV)')
```

```
library(glmnet)

lassoFit <- glmnet(xtrain,ytrain, alpha=1, lambda=lambda.array)
summary(lassoFit)

# Lambdas in relation to the coefficients
plot(lassoFit, xvar = 'lambda', label=T)

# Goodness of fit
plot(lassoFit, xvar = 'dev', label = T)

# Predicted Values
y_predicted_lasso <- predict(lassoFit, s=min(lambda.array), newx = xtest)

# SSE, SST
sst <- sum((ytest - mean(ytest))^2)
sse <- sum((y_predicted_lasso - ytest)^2)

rsquare_lasso <- 1 - (sse/sst)
# MSE
MSE_lasso = (sum((y_predicted_lasso - ytest)^2) / length(y_predicted_lasso))
MSE_lasso

rmse_lasso <- sqrt(MSE_lasso)
```