Machine Learning

Nada

2025-01-30

```r
library(e1071)    # For SVM
library(caret)    # For Confusion Matrix

## Loading required package: ggplot2

## Loading required package: lattice

library(caTools)  # For Data Splitting
library(ggplot2)
library(randomForest)

## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

Machine learning:

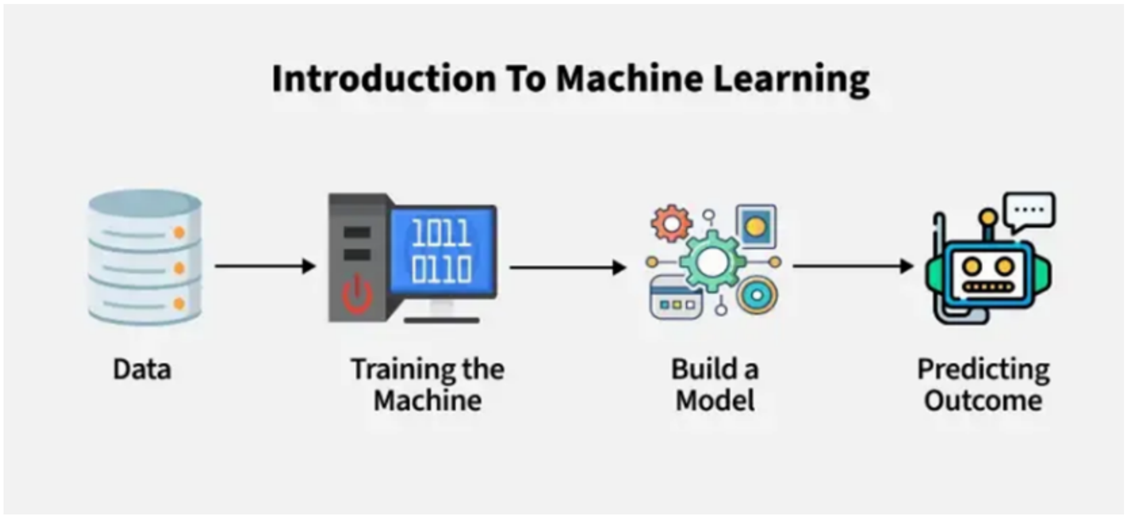
Machine learning is the process in which the developer tries to make the computers mimic the ability of the human brain to learn from experience.

ML allows computers to learn and make decisions without being explicitly programmed.

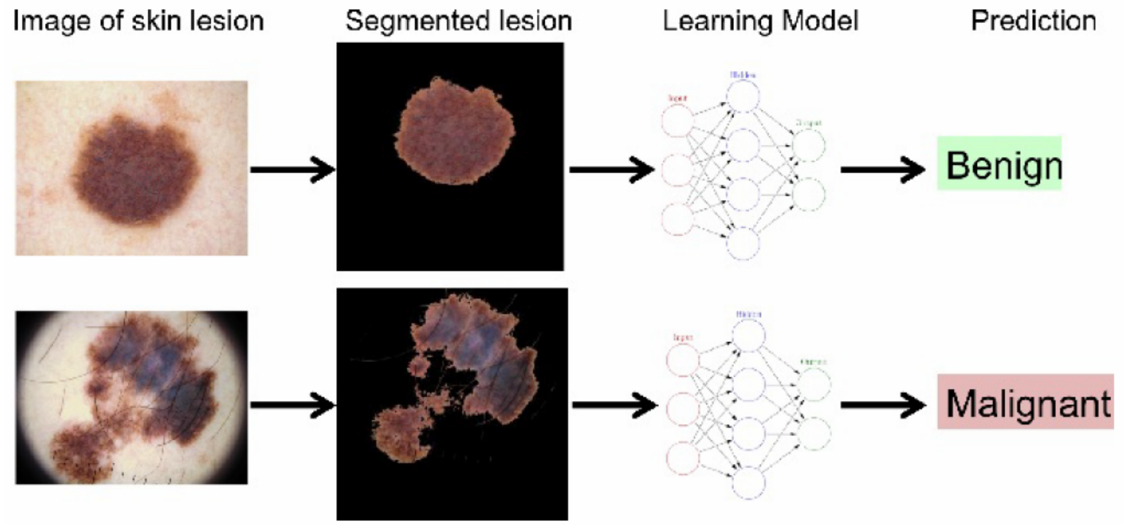
It involves feeding data into algorithms to identify patterns and make predictions on new data.

Machine Learning algorithm learns from data, train on patterns, and solve or predict complex problems beyond the scope of traditional programming.

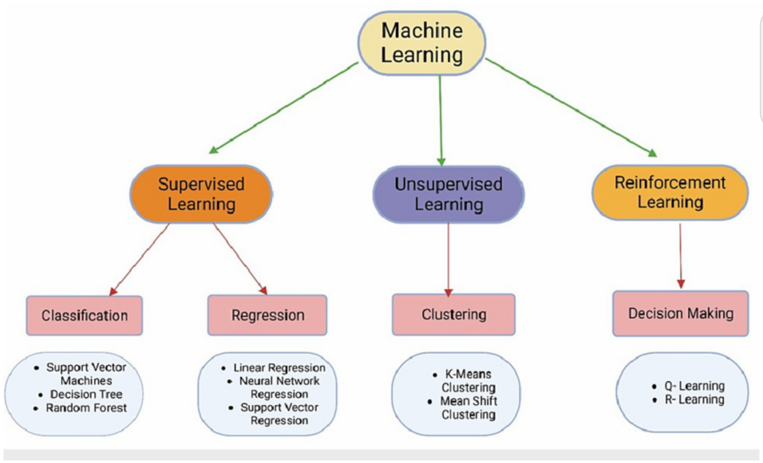
Machine learning is used in various applications, including image and speech recognition, healthcare, Social media platforms and natural language processing.

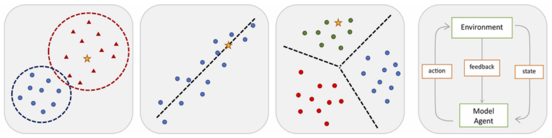


In biology:



Machine learning algorithms:

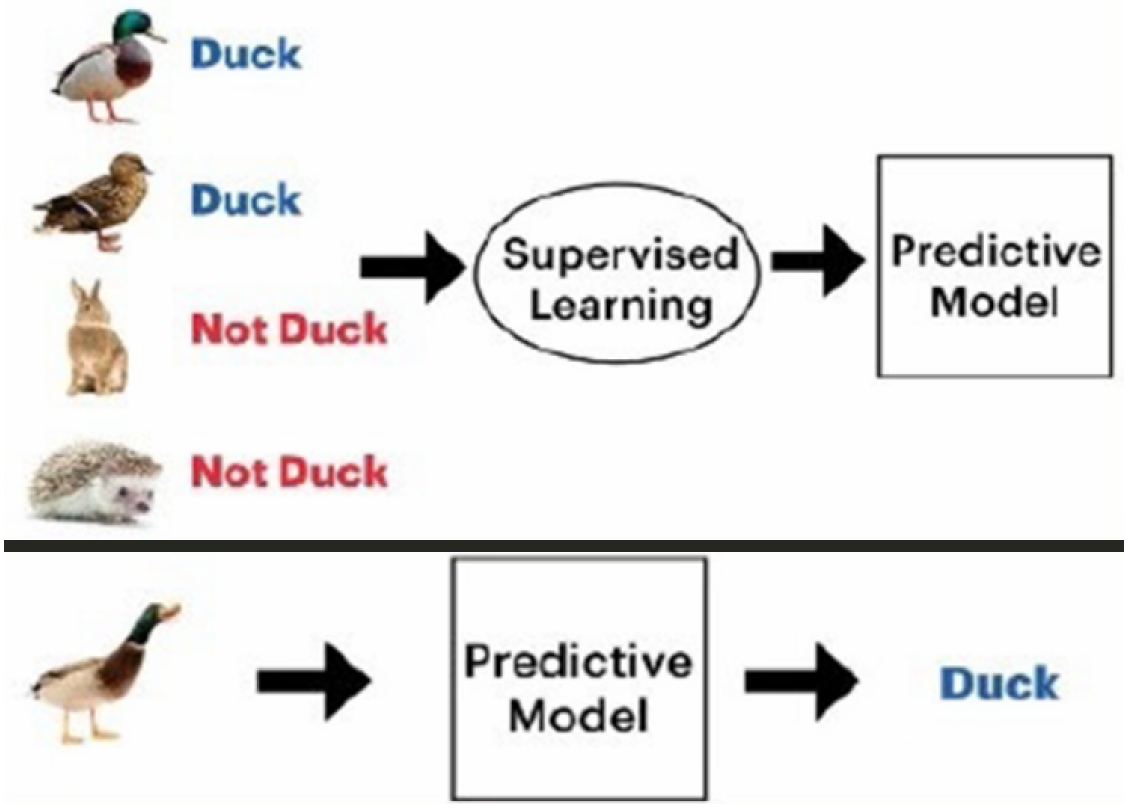




Supervised learning (SML):

the learning algorithm is presented with labelled example inputs, where the labels indicate the desired output.

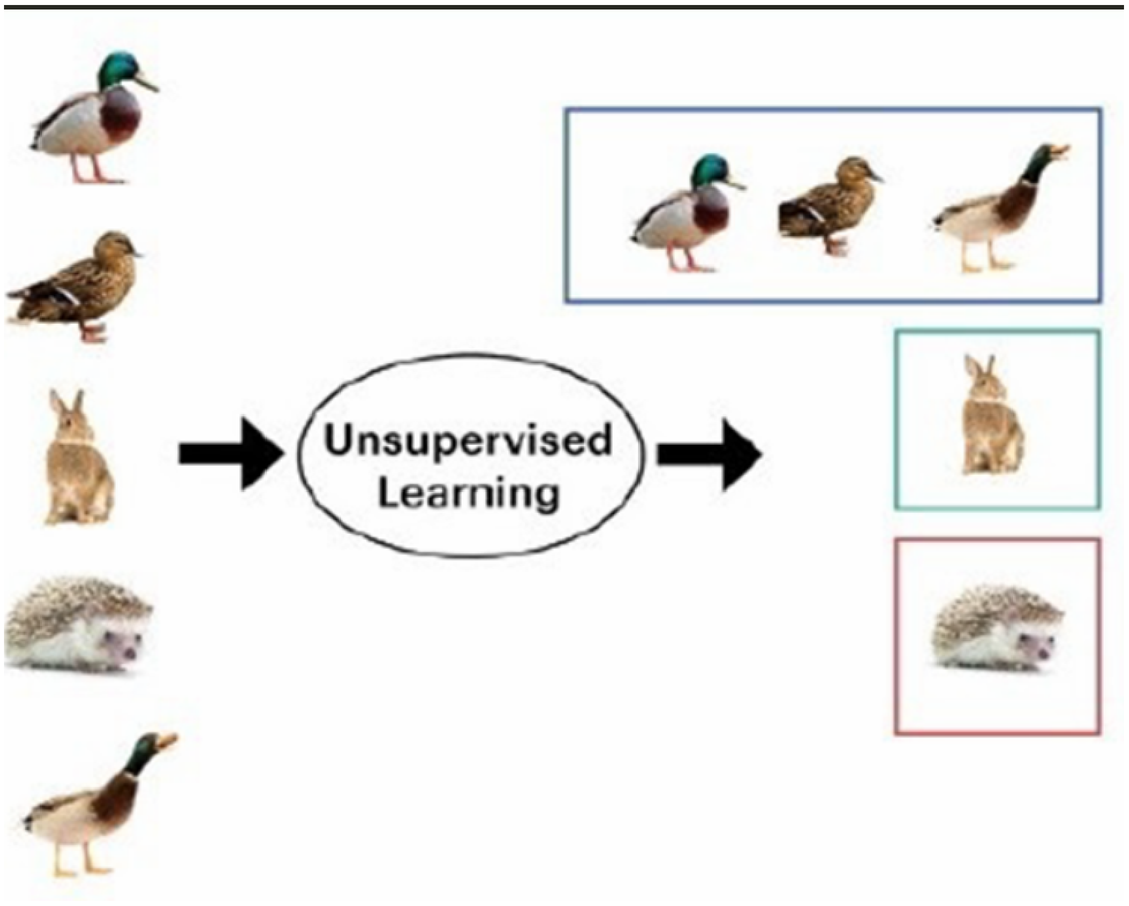
SML itself is composed of classification and regression.



Unsupervised learning (UML):

no labels are provided, and the learning algorithm focuses on detecting structure in unlabeled input data.

UML is composed of clustering and Dimensionality Reduction.



Classification Vs. Clustering:

Although both techniques have certain similarities, the difference lies in the fact that classification uses predefined classes in which objects are assigned, while clustering identifies similarities between objects, which it groups according to those characteristics in common and which differentiate them from other.

Reinforcement learning:

the learning algorithm performs a task using feedback from operating in a real or synthetic environment to make a sequence of decisions.

In this course, we will cover

- Regression (Linear and Logistic)
- Classification (Support vector machines, and Random Forest)
- Clustering (K-means)
- Dimensionality Reduction (Principle Component Analysis)

Clustering:

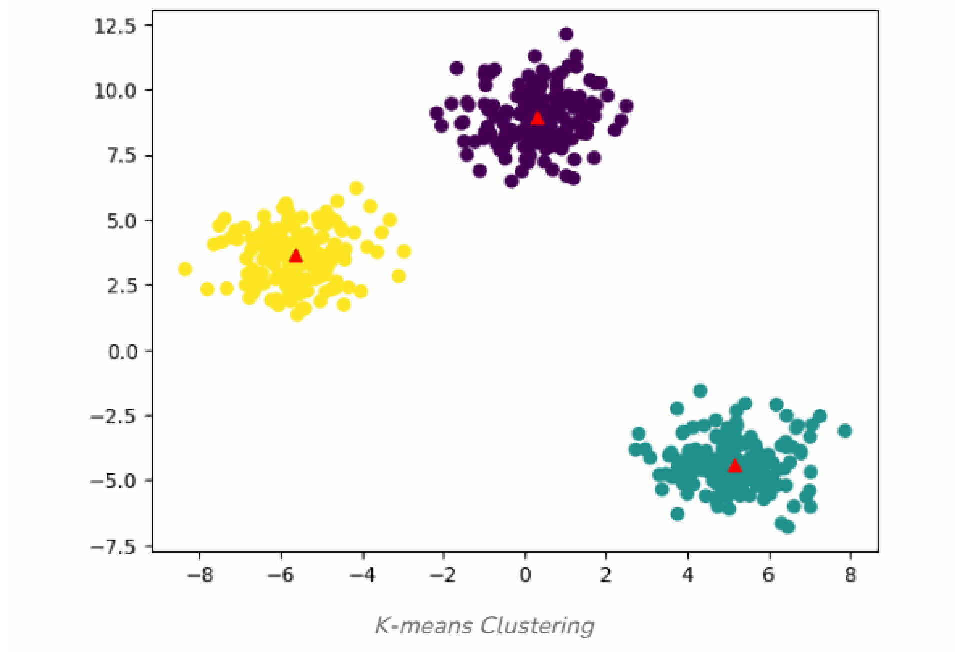
K-means:

The theory of K-means is to cluster the points according to the distance.

The algorithm will categorize the items into k groups or clusters of similarity.

The algorithm works as follows:

1. Choose K (number of clusters)
2. Initialize Centroids by Randomly select K points as the initial cluster centers.
3. Assign Each Point to a Cluster
   - For each data point, calculate the distance to each centroid.
   - Assign it to the closest centroid.
4. **U**pdate Centroids
   - Compute the new centroid for each cluster by taking the average of all points in that cluster.
5. Repeat Until Convergence
   - Reassign points to the closest centroids.
   - Update centroids again.
   - Stop when centroids don’t change or a set number of iterations is reached.



You work in a research lab and have been given an excel file including the rates of exhaustion and headaches as well as the task of predicting the positive COVID cases and the Control cases.

```
set.seed(1)
Data <- rbind(matrix(rnorm(100, sd = 0.3), ncol = 2),
matrix(rnorm(100, mean = 1, sd = 0.3), ncol = 2))
colnames(Data) <- c("tiredness rate", "headache rate")
#View the data
head(Data)

##      tiredness rate headache rate
## [1,]    -0.18793614     0.1194318
## [2,]     0.05509300    -0.1836079
## [3,]    -0.25068858     0.1023359
## [4,]     0.47858424    -0.3388089
## [5,]     0.09885233     0.4299071
## [6,]    -0.24614052     0.5941200

cl <- kmeans(Data, 2)
cl

## K-means clustering with 2 clusters of sizes 49, 51
##
## Cluster means:
##   tiredness rate headache rate
## 1     0.02149367    0.02121248
## 2     0.94443633    1.01712793
##
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 2 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
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 6.525480 8.392416
##  (between_SS / total_SS =  75.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"          "iter"          "ifault"
```

Cluster means Interpretation:

Cluster 1 (Low Symptoms Group)

The average tiredness rate is 0.0215 (very low).

The average headache rate is 0.0212 (also very low). T

his cluster likely represents individuals who report almost no symptoms.

Cluster 2 (High Symptoms Group)

The average tiredness rate is 0.9444 (high).

The average headache rate is 1.0171 (also high).

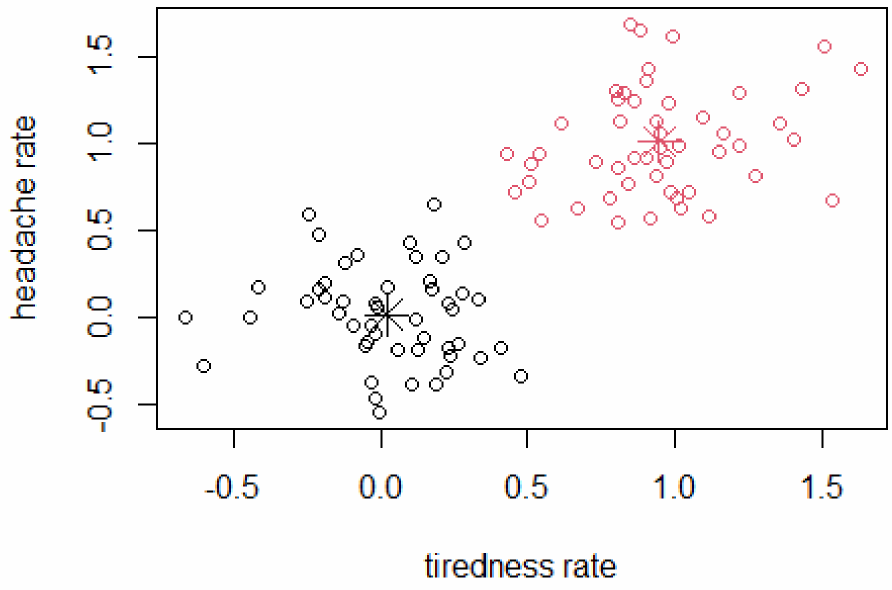
This cluster likely represents individuals who experience strong symptoms.

75.5% BCSS / Total SS Suggests that the clustering is **fairly strong** (clusters are well-separated).

Available components:

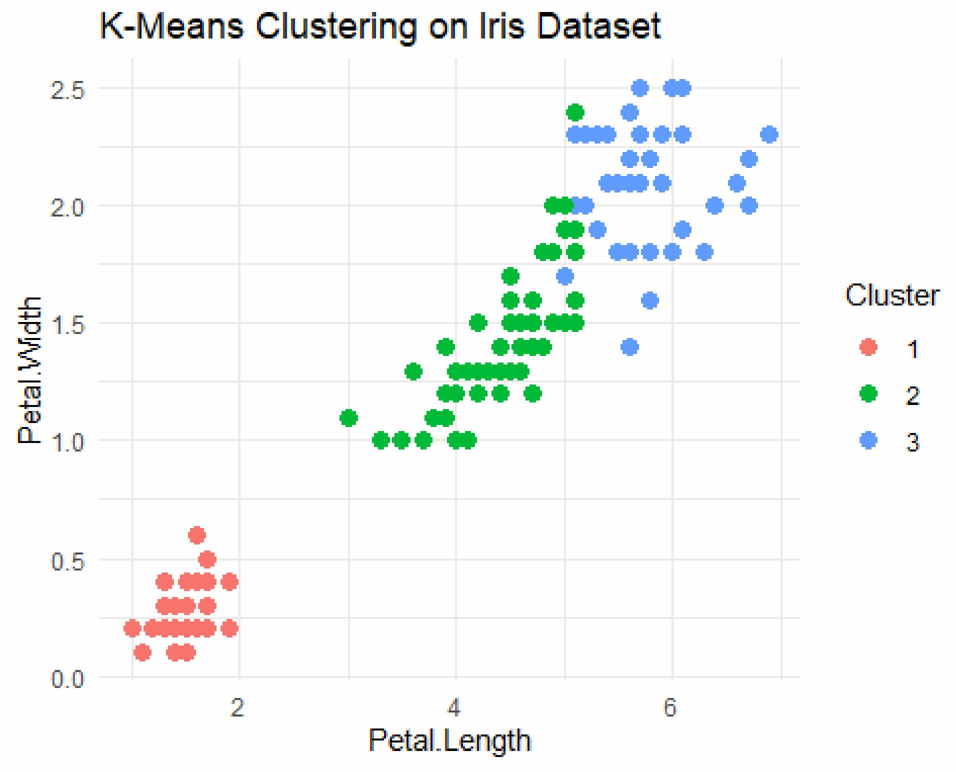
`cluster`	A vector indicating the cluster assignment for each data point.
`centers`	A matrix containing the coordinates of the cluster centers (centroids).
`totss`	Total sum of squares (TSS), which represents the total variance in the dataset.
`withinss`	A vector containing the within-cluster sum of squares (WCSS) for each cluster (how tightly the points are grouped within each cluster).
`tot.withinss`	The total within-cluster sum of squares (WCSS) (sum of all `withinss` values). A lower value indicates better clustering.
`betweenss`	The between-cluster sum of squares (BCSS), which represents the variance explained by the clustering. Higher `betweenss` means better-separated clusters.
`size`	The number of data points assigned to each cluster.
`iter`	The number of iterations the algorithm ran before convergence.
`ifault`	Error flag (0 means the algorithm converged successfully; other values indicate convergence issues).

```
plot(Data, col = cl$cluster)
points(cl$centers, col = 1:2, pch = 8, cex = 2)
```



.

```
data(iris)
df <- iris[, -5]
head(df)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
## 3          4.7         3.2          1.3         0.2
## 4          4.6         3.1          1.5         0.2
## 5          5.0         3.6          1.4         0.2
## 6          5.4         3.9          1.7         0.4

set.seed(123)
k <- 3
kmeans_result <- kmeans(df, centers = k)

df$Cluster <- as.factor(kmeans_result$cluster)
df$Cluster

##   [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
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3
## [112] 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3
## [149] 3 2
## Levels: 1 2 3

ggplot(df, aes(x = Petal.Length, y = Petal.Width, color = Cluster)) +
  geom_point(size = 3) +
  ggtitle("K-Means Clustering on Iris Dataset") +
  theme_minimal()
```



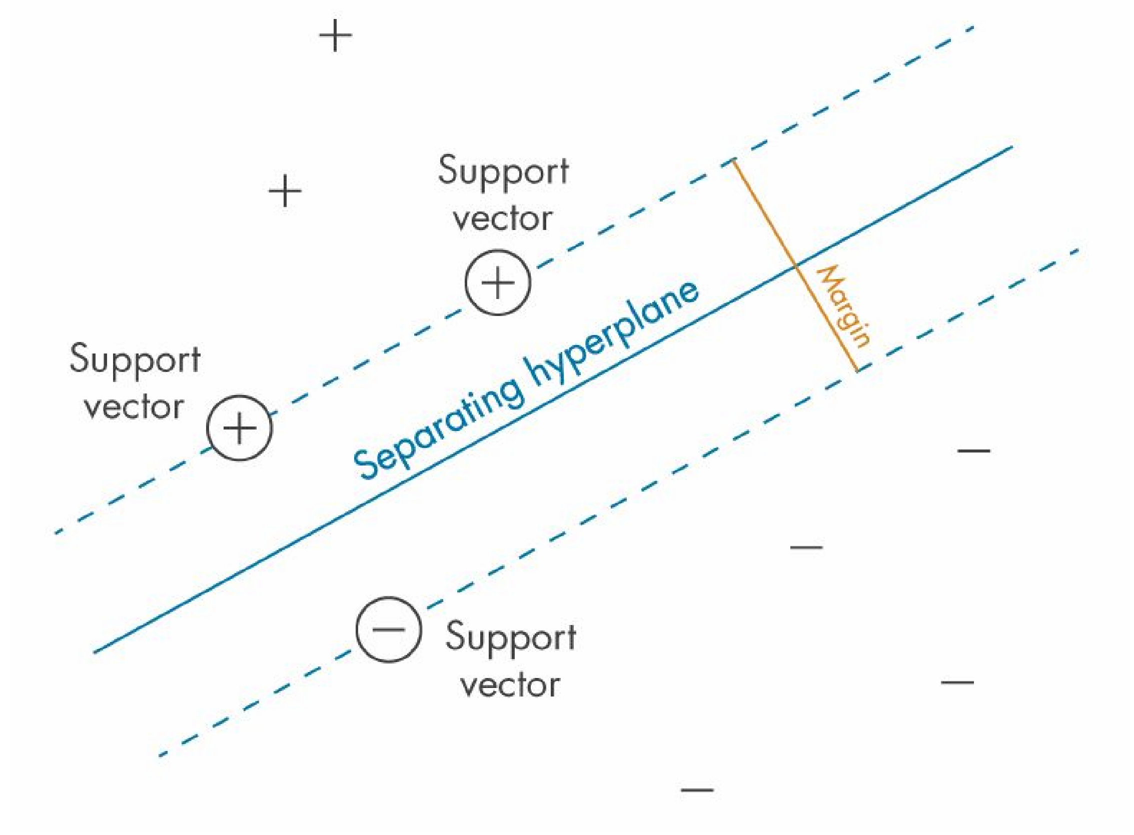
```
table(iris$Species, df$Cluster)

##
##               1  2  3
##   setosa     50  0  0
##   versicolor  0 48  2
##   virginica   0 14 36
```

Classification:

Support Vector Machine(SVM):

The theory of SVM is to find the best hyper-plane that separate the data into classes.

The algorithm maximizes the margin between the closest points of different classes.



1. Randomly place a hyperplane
   - initially, SVM may start with a random hyperplane.
2. Identify Support Vectors
   - Instead of checking all points, SVM identifies the closest points from each class (these are the support vectors).
   - The hyperplane is adjusted based only on these support vectors.
   - Other points do not affect the final decision boundary!
3. Maximize the Margin
   - The algorithm adjusts the hyperplane to maximize the distance (margin) between the support vectors.
   - Only the support vectors contribute to this optimization—the rest of the data is ignored for margin calculations.

```r
Control.data <- matrix(data = rnorm(50000, mean = 0, sd = 0.7), nrow = 1000, ncol = 50) #1000 * 50
colnames(Control.data) <- rep("Control", 50)
#head(Control.data)

Cancer.data <- matrix(data = rnorm(50000, mean = 4, sd = 3), nrow = 1000, ncol = 50) #1000 * 50
colnames(Cancer.data) <- rep("Cancer", 50)
#head(Cancer.data)

Data.mat <- data.frame(cbind(Control.data, Cancer.data)) #1000 * 100
#head(Data.mat)

rownames(Data.mat) <- paste0("Gene_", seq(1:1000))
#head(Data.mat)

Model.data <- data.frame(t(Data.mat))
#head(Model.data)

#Assign the labels
Model.data$Class.Labels <- ifelse(grepl("Control", rownames(Model.data)), "Control", "Cancer")
Model.data$Class.Labels <- as.factor(Model.data$Class.Labels)
```

```r
#Split the data into training and testing
set.seed(123)
split <- sample.split(Model.data$Class.Labels, SplitRatio = 0.6)
Training.set <- Model.data[split, ]
Test.set <- Model.data[!split, ]
Test = Test.set[,-c(length(Test.set))]
#head(Training.set)

classifier <- svm(formula = Class.Labels ~ .,
                  data = Training.set,
                  type = 'C-classification',
                  kernel = 'linear')
```

kernel Type:

Linear Kernel for linearly separable data.

Radial Basis Function(RBF) kernel for non-linearly separable data and Works well for complex decision boundaries.

Polynomial Kernel When data has curved boundaries that a linear kernel cannot separate.

- If unsure, start with RBF Kernel, as it works well in most cases.

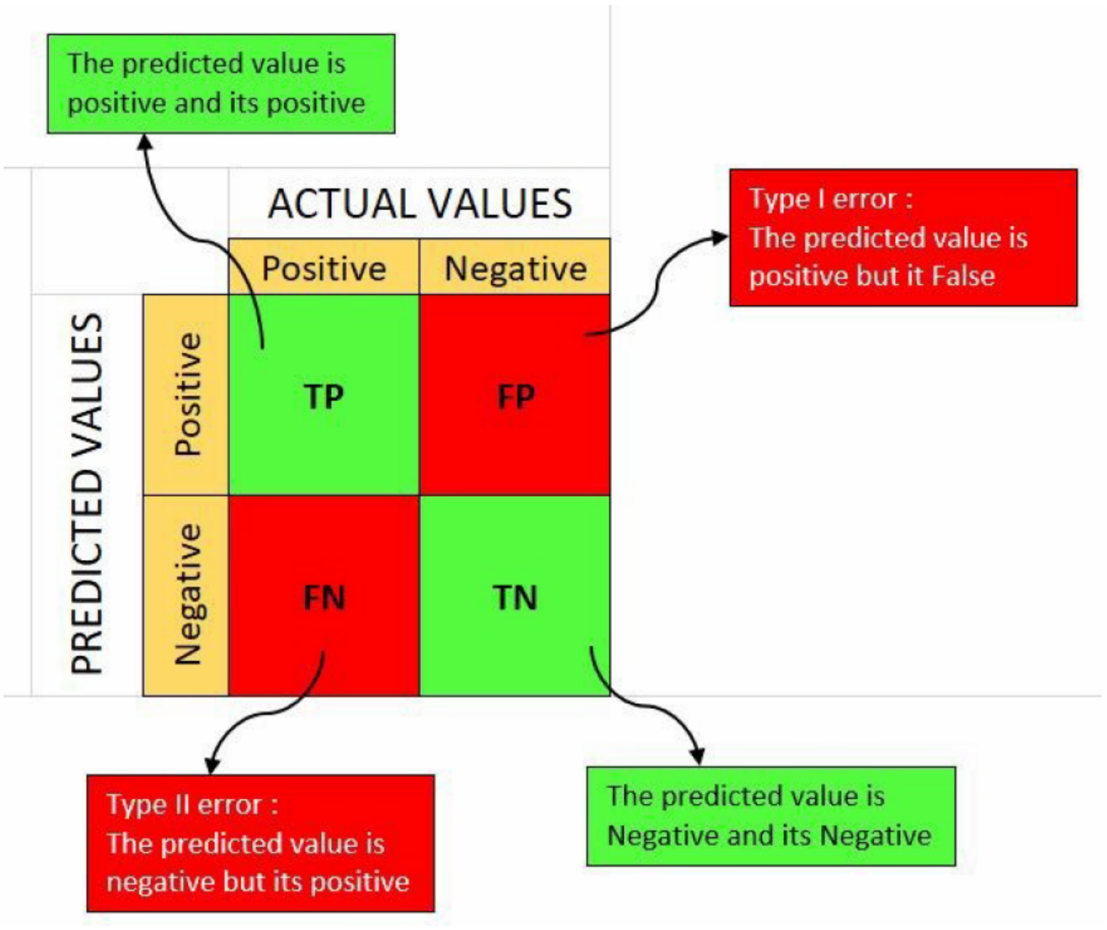
The type of classification or regression depends on the nature of the problem you're trying to solve.

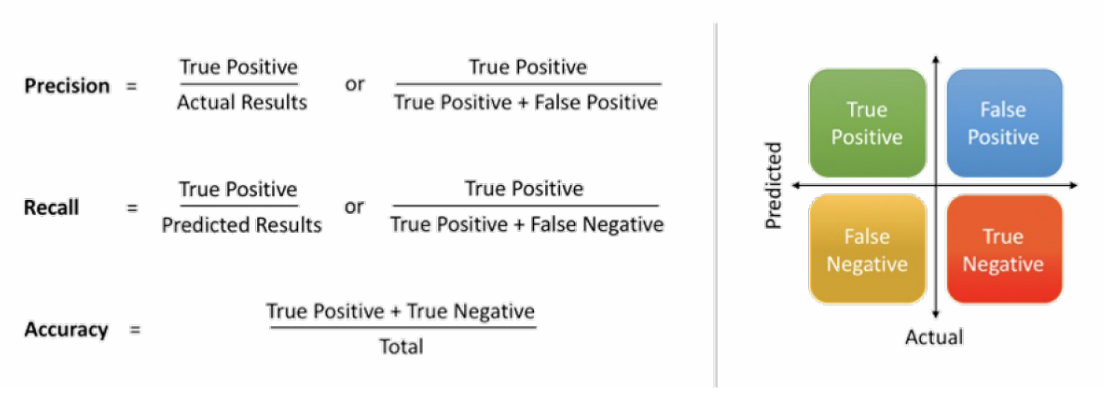
- If you have a classification problem, i.e., discrete label to predict, you can use **C-classification** and **nu-classification**.
- If you have a regression problem, i.e., continuous number to predict, you can use **eps-regression** and **nu-regression**.
- If you only have one class of the data, i.e., normal behavior, and want to detect outliers. **one-classification.

```r
Pred.Labels <- predict(classifier, newdata = Test)

# Generate Confusion Matrix
conf_matrix <- confusionMatrix(Pred.Labels, Test.set$Class.Labels)
conf_matrix
```

```
## Confusion Matrix and Statistics
## 
##             Reference
## Prediction Cancer Control
##    Cancer      20       0
##    Control      0      20
## 
##                Accuracy : 1
##                  95% CI : (0.9119, 1)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 9.095e-13
## 
##                   Kappa : 1
## 
##  Mcnemar's Test P-Value : NA
## 
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.5
##          Detection Rate : 0.5
##    Detection Prevalence : 0.5
##       Balanced Accuracy : 1.0
## 
##        'Positive' Class : Cancer
```

```
##
```


The predicted value is positive and its positive
ACTUAL VALUES
Positive
Negative
PREDICTED VALUES
Positive
Negative
TP
FP
FN
TN
Type I error : The predicted value is positive but it False
Type II error : The predicted value is negative but its positive
The predicted value is Negative and its Negative


Precision = True Positive / Actual Results or True Positive / True Positive + False Positive
Recall = True Positive / Predicted Results or True Positive / True Positive + False Negative
Accuracy = True Positive + True Negative / Total
Predicted
True Positive
False Positive
False Negative
True Negative
Actual

- Accuracy: The proportion of correctly classified samples.
- Sensitivity: The ability to correctly identify positive cases.
- Specificity: The ability to correctly identify negative cases.
- Positive Predictive Value: The proportion of predicted positives that are actual positives.
- Negative Predictive Value: The proportion of predicted negatives that are actual
- Prevalence: The proportion of the dataset that belongs to the positive class.

```r
set.seed(123)
index <- sample(1:nrow(iris), 0.8 * nrow(iris))
train_data <- iris[index, ]
test_data <- iris[-index, ]

# Train SVM model using a radial kernel
svm_model <- svm(Species ~ ., data = train_data, kernel = "radial")
# Print the trained model details
print(svm_model)
```

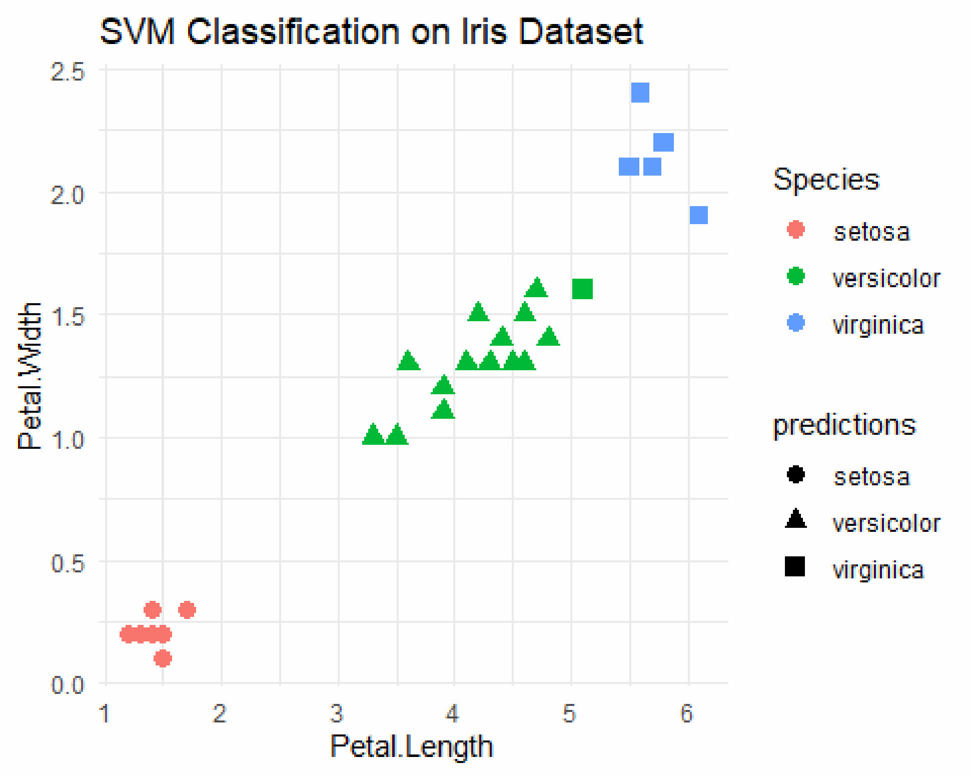
```
##
```

```
## Call:
## svm(formula = Species ~ ., data = train_data, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  46
```

```
# Predict on test data
predictions <- predict(svm_model, test_data)

# Compare predicted vs actual values
confusion_matrix <- confusionMatrix(test_data$Species, predictions)
confusion_matrix
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         14         1
##   virginica       0          0         5
##
## Overall Statistics
##
##                Accuracy : 0.9667
##                  95% CI : (0.8278, 0.9992)
##     No Information Rate : 0.4667
##     P-Value [Acc > NIR] : 4.148e-09
##
##                   Kappa : 0.9464
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           0.8333
## Specificity                 1.0000            0.9375           1.0000
## Pos Pred Value              1.0000            0.9333           1.0000
## Neg Pred Value              1.0000            1.0000           0.9600
## Prevalence                  0.3333            0.4667           0.2000
## Detection Rate              0.3333            0.4667           0.1667
## Detection Prevalence        0.3333            0.5000           0.1667
## Balanced Accuracy           1.0000            0.9688           0.9167
```

```
# Scatter plot of actual vs predicted classes
ggplot(test_data, aes(x = Petal.Length, y = Petal.Width, color = Species, shape = predictions)) +
  geom_point(size = 3) +
  ggtitle("SVM Classification on Iris Dataset") +
  theme_minimal()
```



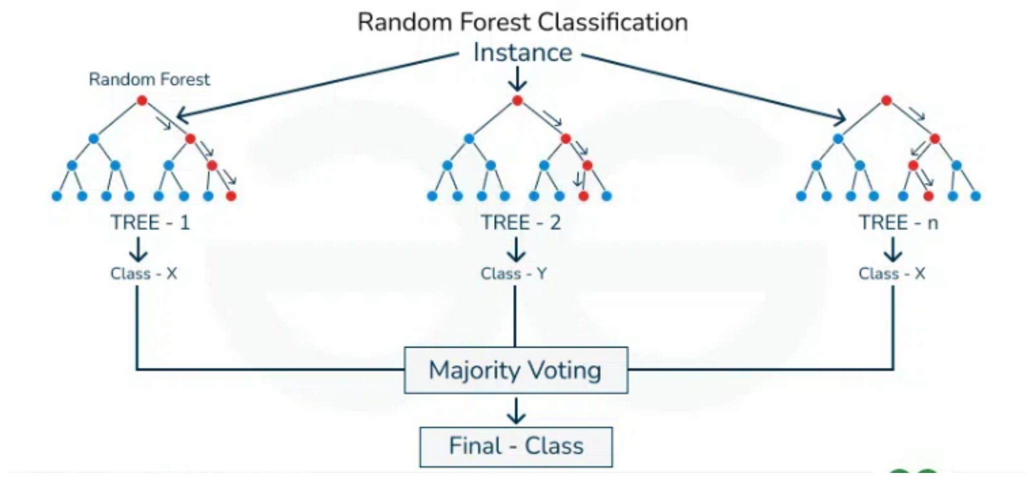
Random Forest:

How Does Random Forest Work?

1. Create Multiple Decision Trees
   - Instead of using a single decision tree, Random Forest creates many decision trees.
   - Each tree is trained on a random subset of the data.
   - Each tree makes its own prediction.
2. Make Predictions
   - the model takes a majority vote (most common class wins).
3. Why Does This Work?
   - By averaging multiple trees, Random Forest reduces overfitting (unlike a single decision tree, which can be too specific).
   - Since each tree sees a different subset of the data, errors and noise are reduced.

Imagine you want to decide who the greatest basketball player is.

- Instead of asking just one expert, you ask 100 different experts.
- Each expert gives their opinion.
- The final answer is the most voted player.



```
# Set random seed to make results reproducible:
set.seed(17)
data("iris")
# Calculate the size of each of the data sets:
split <- sample.split(iris$Species, SplitRatio = 0.6)
# Assign the data to the correct sets
training <- iris[split,]
test <- iris[!split,]
head(training)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
## 7          4.6         3.4          1.4         0.3  setosa
## 8          5.0         3.4          1.5         0.2  setosa
```

```
head(test)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 2           4.9         3.0          1.4         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 13          4.8         3.0          1.4         0.1  setosa
## 14          4.3         3.0          1.1         0.1  setosa
## 15          5.8         4.0          1.2         0.2  setosa
```

```
# Perform training:
rf_classifier = randomForest(Species ~ ., data=training,
              ntree=15, #The model was built using 15 decision trees.
              importance=TRUE)
rf_classifier
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = training, ntree = 15,      importance = TRUE)
##               Type of random forest: classification
##                     Number of trees: 15
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 5.56%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         30          0         0  0.00000000
## versicolor      0         28         2  0.06666667
## virginica       0          3        27  0.10000000
```

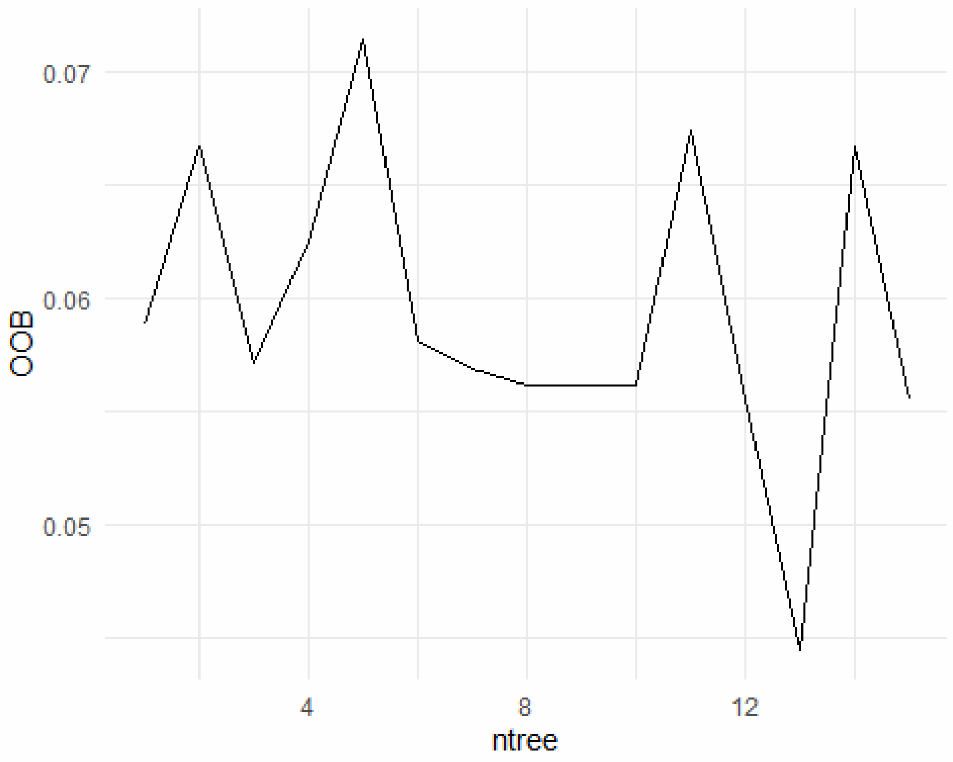
No. of Variables Tried at Each Split At each node, two features were randomly selected to determine the best split.

`importance = TRUE` The model calculates **feature importance** (we can analyze which features are most important for classification).

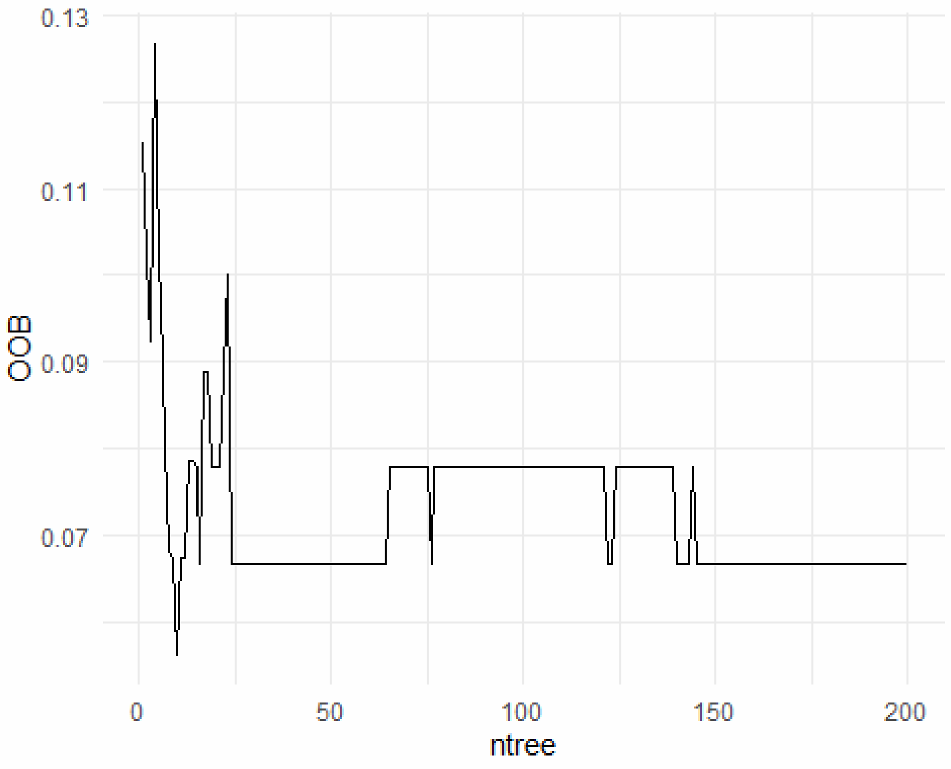
OOB Error

- In Random Forest, each tree is trained using **bootstrap sampling** (random subsets of data).
- The remaining data (out-of-bag samples) are used to estimate error.
- The **OOB error rate of 5.56%** means the model misclassified **5.56% of unseen training samples**.
- This is a good indication of how well the model might generalize to new data.

```
x <- data.frame(rf_classifier[["err.rate"]])
ggplot(x , aes(x = 1:nrow(x), y = OOB)) + geom_line() +
  theme_minimal()+
  xlab("ntree")
```



```
rf_classifier2 = randomForest(Species ~ ., data=training, ntree=200, importance=TRUE) #ntree = 50 ,
100, 200, 500
rf_classifier2

##
## Call:
##  randomForest(formula = Species ~ ., data = training, ntree = 200,      importance = TRUE)
##               Type of random forest: classification
##                     Number of trees: 200
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6.67%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         30          0         0  0.00000000
## versicolor      0         28         2  0.06666667
## virginica       0          4        26  0.13333333

x <- data.frame(rf_classifier2[["err.rate"]])
ggplot(x , aes(x = 1:nrow(x), y = OOB)) + geom_line() +
  theme_minimal()+ xlab("ntree")
```



```
prediction_for_table <- predict(rf_classifier,test[,-5])

confusion_matrix <- confusionMatrix(test[,5],prediction_for_table)
confusion_matrix

## Confusion Matrix and Statistics
##
##                Reference
## Prediction   setosa versicolor virginica
##   setosa         20          0         0
##   versicolor      0         19         1
##   virginica       0          0        20
##
## Overall Statistics
##
##                Accuracy : 0.9833
##                  95% CI : (0.9106, 0.9996)
##     No Information Rate : 0.35
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.975
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           0.9524
## Specificity                 1.0000            0.9756           1.0000
## Pos Pred Value              1.0000            0.9500           1.0000
## Neg Pred Value              1.0000            1.0000           0.9750
## Prevalence                  0.3333            0.3167           0.3500
## Detection Rate              0.3333            0.3167           0.3333
## Detection Prevalence        0.3333            0.3333           0.3333
## Balanced Accuracy           1.0000            0.9878           0.9762
```

Cohen's kappa Vs. Accuracy

Cohen's Kappa is used in classification problems to measure how well a model's predictions agree with the actual labels, while adjusting for agreement that might happen by chance.

Why Not Just Use Accuracy?

Accuracy can be misleading when classes are imbalanced.

For example, imagine a model predicting disease (Yes/No) where 95% of people are healthy and only 5% have the disease.

A dumb model that always predicts "No" would be 95% accurate, but it's useless! Cohen's Kappa adjusts for this and gives a fairer evaluation.

The Kappa statistic (or value) is a metric that compares an Observed Accuracy with an Expected Accuracy

Metric	**Meaning**
Observed Accuracy	How often the model is correct
Expected Accuracy	How often it would be correct **by chance**
Cohen's Kappa	Adjusted accuracy that removes random agreement

KAPPA function:

Actual / Predicted	Apple (A)	Banana (B)	Cherry (C)	Total
Apple (A)	40	5	5	50
Banana (B)	10	30	10	50
Cherry (C)	5	10	35	50

Total samples = 150

Step 1: Compute (Observed Agreement)

- correct prediction / total sum = (40+30+35) / 150 = 0.70

Model correctly predicted 70% of the cases

Step 2: Compute (Chance Agreement)

1. Compute class probabilities

- Apple :

  actual = 50/150 = 0.33

  predicted = 55/150 = 0.37

- Banana :

  actual = 0.33

  predicted = 0.30

- Cherry :

  actual = 0.33

  predicted = 0.33

2. Compute Chance Agreement using probability multiplication

   $(0.33\times0.37)+(0.33\times0.30)+(0.33\times0.33) = 0.33$

Chance agreement is 33%

Step 3: Compute Cohen's Kappa

(Observed Agreement - Chance Agreement) /(1 - Chance Agreement) =

(0.70 - 0.33) / (1 - 0.33) = 0.55

Interpretation

Kappa Value	**Agreement Level**
< 0	Worse than random
0 - 0.20	Slight agreement
0.21 - 0.40	Fair agreement
0.41 - 0.60	Moderate agreement
0.61 - 0.80	Substantial agreement
0.81 - 1.00	Almost perfect