Machine learning is a sub-field of artificial intelligence (AI) that provides systems the ability to **automatically learn and improve from experience without being explicitly programmed.** It involves showing a large volume of data to a machine so that it can learn and make predictions, find patterns, or classify data. The three machine learning types are supervised, unsupervised, and reinforcement learning.

For the process of **learning (model fitting)** we need to have available **some observations or data (also known as samples or examples)** in order to **explore potential underlying patterns, hidden in our data**. These learned patterns are nothing more than **some functions or decision boundaries.** The outcome you provide the machine is **labeled data**, and the rest of the information you give is used as input features.

Supervised: For the training procedure, the input is a known training data set with **its corresponding labels,** and the learning algorithm produces an inferred function to finally make predictions about some new unseen observations that one can give to the model. The model is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct intended output (ground truth label) and find errors in order to modify itself accordingly (e.g. via back-propagation). This machine learning type got its name because the machine is “supervised” while it's learning, which means that you’re feeding the algorithm information to help it learn.

While supervised learning requires users to help the machine learn, **unsupervised learning** doesn't use the labeled training sets and data. Instead, the machine looks for **less obvious patterns in the data.** This machine learning type is very helpful when you need to **identify patterns and use data to make decisions. Unsupervised learning** studies how systems can infer a function to describe a **hidden structure from unlabeled data.** The **system doesn’t predict the right output,** but instead, **it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.**

Reinforcement learning: Reinforcement learning is the closest machine learning type to how humans learn. The algorithm or agent used, learns by interacting with its environment and getting a positive or negative reward.
This family of models consists of algorithms that use the estimated errors as rewards or penalties. If the error is big, then the penalty is high and the reward low. If the error is small, then the penalty is low and the reward high.

Regression and Classification algorithms are **Supervised Learning** algorithms. Both the algorithms are used for **prediction in Machine learning and work with the labeled datasets.** But the difference between both is how they are used for different machine learning problems.

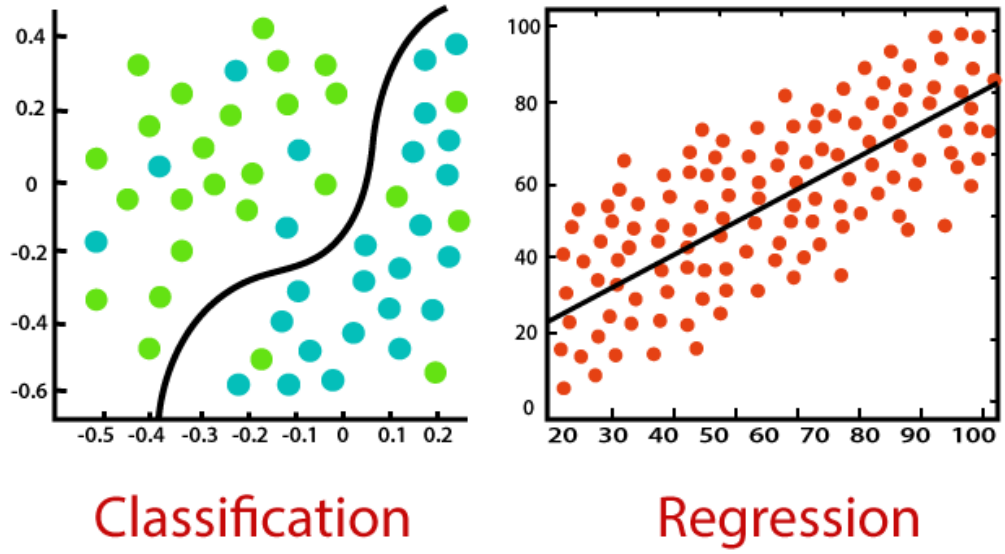
The main difference between Regression and Classification algorithms that **Regression algorithms are used to predict the continuous values such as price, salary, age, etc. and Classification algorithms are used to predict/Classify the discrete values such as Male or Female, True or False, Spam or Not Spam, etc.**

Regression Algorithm Classification Algorithm

In Regression, the output variable must be of continuous nature or real value.
In Classification, the output variable must be a discrete value.

In Regression, we try to find the **best fit line,** which can predict the output more accurately.
In Classification, we try to find the **decision boundary**, which can divide the dataset into different classes.

Regression algorithms can be used to solve the regression problems such as Weather Prediction, House price prediction, etc.
Classification Algorithms can be used to solve classification problems such as Identification of spam emails, Speech Recognition, Identification of cancer cells, etc.
The regression Algorithm can be further divided into Linear and Non-linear Regression.
The Classification algorithms can be divided into Binary Classifier and Multi-class Classifier.



Types of ML Classification Algorithms:

Classification Algorithms can be further divided into the following types:

- **Logistic Regression**
- K-Nearest Neighbours
- Support Vector Machines
- Kernel SVM
- Naïve Bayes
- Decision Tree Classification
- Random Forest Classification

Types of Regression Algorithm:

- **Simple Linear Regression**
- Multiple Linear Regression
- Polynomial Regression
- Support Vector Regression
- Decision Tree Regression
- Random Forest Regression

Regression

The R programming language also provides **functions to estimate statistical models.** One of the most commonly used model types is **linear regression.** Using the **lm and summary functions in R,** we can estimate and evaluate these models.

Linear Regression establishes a relationship between a Dependent variable i.e. Y and one or more Independent variables i.e X**, using a best fit straight line known as Regression Line.** The equation of this regression line can then be used to predict the value of 'Y' for any given 'X'.

Y=f(x)

Y=aX+b

Dependent Variable (Target) : Continuous

Independent Variable(Predictor(s)): Continuous/Discrete

Simple linear regression involves one target (Y) and one predictor (X). This demo performs simple linear regression using **Least Squares Method** to find a regression line that shows trend in the data i.e. relationship between X and Y.

The equation of regression line in slope-intercept form is:

Y = mX + c , where m= slope of straight line

c= Y-intercept

Because we are fitting a linear model, we assume that the relationship really is linear, and that the errors, or residuals, are simply random fluctuations around the true line. We assume that the variability in the response doesn't increase as the value of the predictor increases. This is the assumption of equal variance.

```
#Regression and classification in RStudio on iris dataset
View(iris)
plot(iris[,1:4])
cor(iris[ , 1:4])     # Return correlation matrix
#0.96
Y<- iris[,"Petal.Width"] # select Target attribute
X<- iris[,"Petal.Length"] # select Predictor
head(X)
head(Y)
Species_col=as.numeric(iris$Species)
plot(Y~X,col=Species_col)
model1<- lm(Y~X)
model1 # provides regression line coefficients only i.e. slope and y-intercept
abline(model1, col="blue", lwd=3) #add regression line to scatter plot to see relationship
between X and Y
#Perform prediction
p1<- predict(model1,data.frame("X"=6))
p1

#The predicted value of Petal.Width is 2.1 when Petal.Length= 6
#Exercise (Will check on Monday-second hour)
#Do the same with “Sepal.Length and Sepal.Width”:
```

There are three error metrics that are commonly used for **evaluating and reporting the performance of a regression model;** they are:

- Mean Squared Error (MSE).
- Root Mean Squared Error (RMSE).
- Mean Absolute Error (MAE)

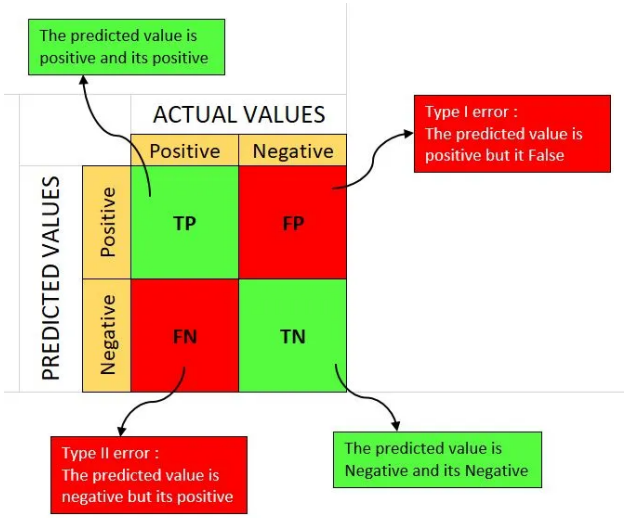
Classification:

```
#---------Classification----------
ir_data=iris
str(ir_data)
levels(ir_data$Species)
sum(is.na(ir_data))

ir_data<-ir_data[1:100,] #taking only two species
#set.seed(100)
samp<-sample(1:100,80)
ir_train<-ir_data[samp,]
str(ir_train)
ir_test<-ir_data[-samp,]
str(ir_test)
y<-ir_train$Species
x<-ir_train$Sepal.Width
plot(x,y)
glfit<-glm(y~x, family = 'binomial') #fits a generalized linear model (GLM) using the binomial family. It assumes that you have a dependent variable y and an independent variable x in your dataset.
# summary(glfit)
#single prediction
newdata<- data.frame(x=2.0)
predicted_val<-predict(glfit, newdata, type="response")
predicted_val
predict_reg <- ifelse(predicted_val >0.5, print("Versicolor"), print("Setosa"))
predict_reg
#Do prediction on testing set
newdata <- data.frame(x=ir_test$Sepal.Width)
predicted_val <- predict(glfit, newdata, type="response")
predicted_val

#Tabular view of actual and predicted values
prediction <- data.frame(ir_test$Sepal.Width, ir_test$Species, predicted_val, Predicted_Species=ifelse(predicted_val >0.5, print("Versicolor"), print("Setosa")))
prediction[,]
View(prediction)
```

Classification evaluation:

Accuracy: The Accuracy score is calculated by dividing the number of correct predictions by the total predictions.

#----------Confusion Matrix—---------



#----------Confusion matrix—---------

```
predicted <- c(1, 1, 1, 0, 0)
actual <- c(1, 0, 1, 1, 1)
xtab <- table(predicted, actual) #will give you confusion matrix
```

#Install caret

library(caret)

```
cm <- caret::confusionMatrix(xtab)
cm
```

#Confusion matrix for iris predictions

```
#predict_reg <- ifelse(predicted_val >0.5, print("Versicolor"), print("Setosa"))
predict_reg <- ifelse(predicted_val >0.5, 2, 1)
table(as.numeric(ir_test$Species), predict_reg)
predict_reg
```

#Performance of the model

```
#Accuracy = (TP+TN)/(TP+FP+TN+FN)
xtab <- table(as.numeric(ir_test$Species), predict_reg)
```

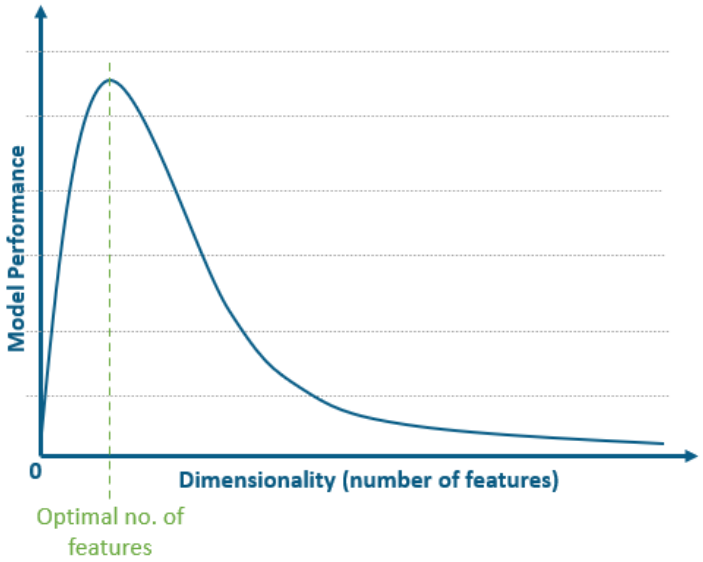
```
xtab
acc=(as.numeric(xtab[1,1])+as.numeric(xtab[2,2]))/(as.numeric(xtab[1,1])+as.numeric(xtab[1,2])+as.numeric(xtab[2,1])+as.numeric(xtab[2,2]))*100
acc
#----------------------Other metrics----
#Accuracy = (TP+TN)/(TP+FP+TN+FN)
#Precision = TP / (TP + FP)
#Recall or Sensitivity = TP / (TP + FN)
#F1 = 2 * (Precision * Recall) / (Precision + Recall)
#Precision — Out of all the examples that are predicted as positive, how many are really positive?
#Recall — Out of all the positive examples, how many are predicted as positive?
#Specificity — Out of all the people that do not have the disease, how many got negative results?
#-----------------
#Using package
library(Metrics)
Metrics::accuracy(as.numeric(ir_test$Species), predict_reg)
# to calculate precision
Metrics::recall(as.numeric(ir_test$Species), predict_reg)
# to calculate precision
Metrics::precision(as.numeric(ir_test$Species), predict_reg)
# to calculate f1_score
Metrics::f1(as.numeric(ir_test$Species), predict_reg)


#--------------------------------------
#confusion matrix
xtab <- table(as.numeric(ir_valid$Species), predict_reg)
library(caret)
cm <- caret::confusionMatrix(xtab)
cm
```

Curse of Dimensionality:

This term describes how the **increase in input data dimensions** results in an **exponential increase in computational expense and efforts r**equired for processing and analyzing that data.

What is the Curse of Dimensionality?

The curse of dimensionality basically refers to the difficulties a machine learning algorithm faces when working with data in the higher dimensions that did not exist in the lower dimensions. This happens because when you **add dimensions (features), the minimum data requirements also increase rapidly.**



Hence, to remove the curse afflicting your model, you might need to put your data on a diet – i.e., **reduce its dimensions through feature selection and feature engineering techniques.**

How is Dimensionality Reduction done?

Several techniques can be employed for dimensionality reduction depending on the problem and the data. These techniques are divided into two broad categories:

Feature Selection: Choosing the **most important features** from the data

Feature Extraction: Combining features to create new super features.

#----------

Why is Dimensionality Reduction necessary?

Avoids overfitting – the fewer assumptions a model makes, the simpler it will be.

Easier computation – the lesser the dimensions, the faster the model trains.

Improved model performance – removes redundant features and noise, lesser misleading data improves model accuracy.

Lower dimensional data requires less storage space.

Lower dimensional data can work with other algorithms that were unfit for larger dimensions.

With the increase in the data dimensions, your model –

- would also **increase in complexity.**
- would become **increasingly dependent on the data** it is being trained on.

This leads to **overfitting** of the model, so even though the **model performs really well on training data, it fails drastically on any real data.**

#Doubts

```
fun1 <- function() {
  x <- 10          #local scope
  print(x)
}
x <- 5    #  (global) scope
fun1()
print(x)
```

```
fun1 <- function() {
  x  <<-  10          #local scope
  print(x)
}
x <- 5    #  (global) scope
fun1()
print(x)
```

```
#Columnwise binding in dataframe
empid <- c(1:4)
empname <- c("Sam","Rob","Max","John")
empdept <- c("Sales","Marketing","HR","R & D")
emp.data <- data.frame(empid,empname,empdept)
empid <- c(1:3)
deptname <- c("Fran","Ton","Eri")
salary <- c(20000,30000,40000)
emp.newdata <- data.frame(empid, deptname,salary)
df_merge <- merge(emp.data,emp.newdata,by = "empid")
# print("Merged DataFrame")
print(df_merge)

empid <- c(1:4)
empname <- c("Sam","Rob","Max","John")
empdept <- c("Sales","Marketing","HR","R & D")
emp.data <- data.frame(empid,empname,empdept)
  empid <- c(3:5)
  deptname <- c("Fran","Ton","Eri")
  salary <- c(20000,30000,40000)
  emp.newdata <- data.frame(empid, deptname,salary)
df_merge <- merge(emp.data,emp.newdata,by = "empid")
# print("Merged DataFrame")
print(df_merge)
```