UNIT-III REGRESSION AND BLACK BOX METHODS

Forecasting numerical data – Understanding regression – predicting medical expenses using linear regression -Understanding regression trees and model trees – estimating the quality of wines with regression trees and model trees– Neural Networks and SVM – Understanding neural networks – modelling the strength of concrete with ANNs – Understanding Support Vector Machines – performance OCR with SVMs.

Regression

Regression in machine learning is a supervised learning technique used for predicting a continuous outcome or dependent variable (also called the target or response variable) based on one or more independent variables (also called features or predictors). It is primarily used for solving regression problems, where the goal is to establish a relationship between the input variables and the output variable, allowing you to make predictions or estimate values for the output variable when new input data is provided. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable. It assumes that the relationship between the independent and dependent variables follow a straight line. Example: Age and Health, Education and Income

Types of Regression:

- Linear Regression: Assumes a linear relationship between the independent variables and the dependent variable. Simple linear regression deals with one independent variable, while multiple linear regression handles multiple independent variables.
- Polynomial Regression: Extends linear regression by considering polynomial functions of the independent variables.
- Ridge Regression and Lasso Regression: Used for regularization to prevent overfitting in linear regression by adding penalty terms to the loss function.
- Support Vector Regression (SVR): Utilizes support vector machines to perform regression tasks.
- Decision Tree Regression: Employs decision trees to make predictions.
- Random Forest Regression: An ensemble technique that combines multiple decision trees for better accuracy.
- Gradient Boosting Regression: A boosting ensemble method that builds multiple weak models to create a strong predictive model.
- Neural Network Regression: Utilizes artificial neural networks for regression tasks, which can handle complex relationships.

Applications

Regression has a wide range of real-life applications. It is essential for any machine learning problem that involves continuous numbers such as

- Financial forecasting (like house price estimates, or stock prices)
- Sales and promotions forecasting
- Testing automobiles
- Weather analysis and prediction
- Time series forecasting
- Modelling complex relationships among data elements

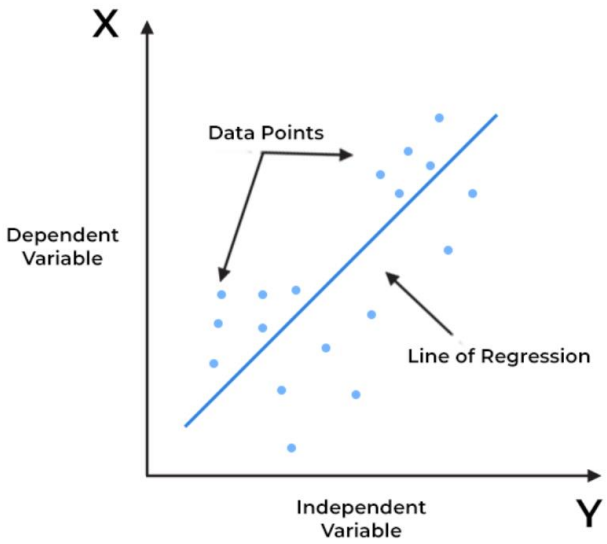
- Quantify causal relationship between event and response
- Identify patterns to forecast future

Linear Regression

Linear regression is a statistical approach to model the relationship between the outcome variable and predictor variable(s) by fitting a linear equation to observed data. It is used to predict the value of a variable based on the value of another variable. A linear regression is a linear approximation of a causal relationship between two variables. It aims to find the best-fitting line that describes the relationship. The line is determined by minimizing the sum of the squared differences between the predicted values and the actual values.

Simple Linear Regression

In a simple linear regression, there is one independent variable and one dependent variable. The model estimates the slope and intercept of the line of best fit, which represents the relationship between the variables. The slope represents the change in the dependent variable for each unit change in the independent variable, while the intercept represents the predicted value of the dependent variable when the independent variable is zero.



Linear regression produces a straight line on the graph. Mathematically

$$y = ax + b$$

where,

- x indicates predictor or independent variable
- y indicates response or dependent variable
- a and b are coefficients

Linear Regression in R

In R, the **lm()** function is used to perform linear regression analysis. The "lm" function stands for "linear model." This function creates the relationship model between the predictor and the response variable.

Syntax

The basic syntax for lm() function in linear regression is −

lm(formula,data)

where

formula is a symbol presenting the relation between x and y.

data is the vector on which the formula will be applied.

Example:

```
data <- data.frame(
  Height = c(160, 165, 155, 170, 175),
  Weight = c(55, 60, 50, 70, 75)
)
model <- lm(Height ~ Weight, data = data)
new_weight <- 68
predicted_height <- predict(model, newdata = data.frame(Weight = new_weight))
cat("Predicted Height for Weight", new_weight, "is:", predicted_height)
```

Output:
Predicted Height for Weight 68 is 169.5345

predict() Function

The predict() function in R is used to make predictions from a fitted statistical model, such as a linear regression model, logistic regression model, or other types of models. It takes a fitted model object and a new data as input and returns predicted values based on the model.

Syntax

The basic syntax for predict() in linear regression is −

predict(object, newdata)

where, object is the formula which is already created using the lm() function
newdata is the vector containing the new value for predictor variable.

Predict the weight of new persons

```
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)
a <- data.frame(x = 170)
result <-  predict(relation,a)
print(result)
```

Ordinary Least Squares Method

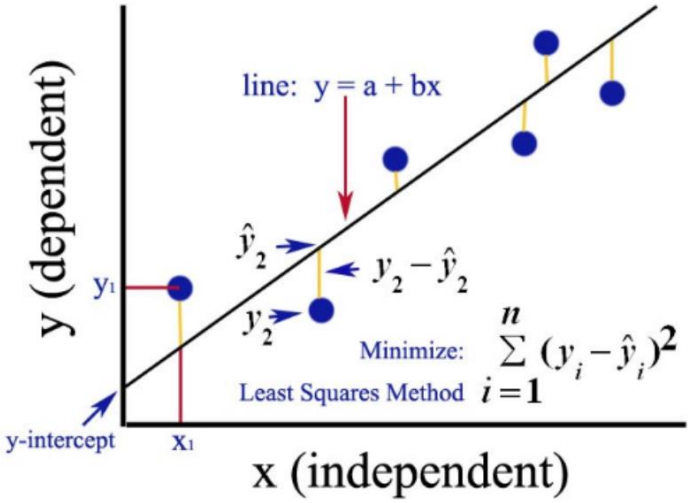
The Ordinary Least Squares (OLS) method is a statistical technique used for fitting a linear regression model to data. It's a fundamental method in statistics and data analysis,

particularly in the context of modelling the relationship between two or more variables. Ordinary least squares (OLS) regression is an optimization strategy that allows you to find a straight line that's as close as possible to your data points in a linear regression model. **Objective:** The primary goal of OLS is to find the best-fitting linear relationship between a dependent variable (response) and one or more independent variables (predictors).

OLS assumes that the relationship between the dependent variable (Y) and the independent variable(s) (X) is **linear.** This means that changes in Y can be explained by changes in X through a linear equation.OLS aims to find the equation of a straight line that minimizes the sum of the squared differences (residuals) between the observed and predicted values of the dependent variable. This line is often referred to as **the regression line.**The equation of the regression line typically takes the form:

$$Y = \beta0 + \beta1*X + \varepsilon$$

Y: Dependent variable (e.g., outcome, response).
X: Independent variable(s) (e.g., predictors).
β0: Intercept (the value of Y when X is zero).
β1: Slope (the change in Y for a one-unit change in X).
ε: Error term (the difference between the observed and predicted values).



The formula for calculating the coefficient β1 that represents the slope of the regression line in Ordinary Least Squares (OLS) linear regression can be given by

$$\beta1 = \Sigma[(xi - \bar{x})(yi - \bar{y})] / \Sigma[(xi - \bar{x})^2]$$

Let's say we have a dataset that represents the relationship between the number of hours studied (x) and the test scores (y) of five students. Here are the data points:

Student	Hours Studied (x)	Test Score (y)
1	2	65
2	3	75
3	4	85
4	5	90
5	6	95

Step 1: Calculate the Mean of x ($\bar{x}$):

$\bar{x} = (2 + 3 + 4 + 5 + 6) / 5 = 20 / 5 = 4$

Now, we have the mean of x, which is 4.

Step 2: Calculate the Difference Between xi and $\bar{x}$:

We calculate the difference between each xi and $\bar{x}$. This shows how far each data point is from the mean of x.

- For Student 1: $xi - \bar{x} = 2 - 4 = -2$
- For Student 2: $xi - \bar{x} = 3 - 4 = -1$
- For Student 3: $xi - \bar{x} = 4 - 4 = 0$
- For Student 4: $xi - \bar{x} = 5 - 4 = 1$
- For Student 5: $xi - \bar{x} = 6 - 4 = 2$

These differences represent how much each student's study hours deviate from the mean study hours.

Step 3: Multiply the Residuals and Differences:

Calculate the product of the residuals ($yi - \bar{y}$) and the differences between xi and $\bar{x}$ ($xi - \bar{x}$). These products tell us how the test scores deviate from their mean ($\bar{y}$) and how the hours studied deviate from their mean ($\bar{x}$) simultaneously.

- For Student 1: $(xi - \bar{x})(yi - \bar{y}) = (-2) * (65 - 82) = 34$
- For Student 2: $(xi - \bar{x})(yi - \bar{y}) = (-1) * (75 - 82) = 7$
- For Student 3: $(xi - \bar{x})(yi - \bar{y}) = (0) * (85 - 82) = 0$
- For Student 4: $(xi - \bar{x})(yi - \bar{y}) = (1) * (90 - 82) = 8$
- For Student 5: $(xi - \bar{x})(yi - \bar{y}) = (2) * (95 - 82) = 26$

Step 4: Summation of the Products:

$\Sigma[(xi - \bar{x})(yi - \bar{y})] = 34 + 7 + 0 + 8 + 26 = 75$

This sum represents the total combined deviation of test scores and hours studied from their respective means for all the students.

Step 5: Define the Difference Between xi and $\bar{x}$ Squared:

This step calculates the squared difference between each xi and $\bar{x}$. It's used to find the sum of squared differences between hours studied and the mean of hours studied.

For Student 1: $(xi - \bar{x})^2 = (-2)^2 = 4$
For Student 2: $(xi - \bar{x})^2 = (-1)^2 = 1$
For Student 3: $(xi - \bar{x})^2 = (0)^2 = 0$
For Student 4: $(xi - \bar{x})^2 = (1)^2 = 1$
For Student 5: $(xi - \bar{x})^2 = (2)^2 = 4$

Step 6: Summation of the Squared Differences:

$\Sigma[(xi - \bar{x})^2] = 4 + 1 + 0 + 1 + 4 = 10$

This sum represents the total sum of squared differences between hours studied and the mean of hours studied for all the students.

Step 7: Calculate β1:

$\beta 1 = \Sigma[(xi - \bar{x})(yi - \bar{y})] / \Sigma[(xi - \bar{x})^2] = 75 / 10 = 7.5$

This shows that, on average, for each additional hour studied (x), the test score (y) is expected to increase by 7.5 points.

Assumptions of OLS:

OLS assumes several conditions, including linearity, independence of errors, constant variance of errors (homoscedasticity), and normality of errors.

Multiple Regression

Multiple regression is another type of regression analysis technique that is an extension of the linear regression model as it uses more than one predictor variables to create the model. Mathematically,

$$y = a + b1x1 + b2x2 + ...bnxn$$

- **y** is the response variable.
- **a, b1, b2...bn** are the coefficients.
- **x1, x2, ...xn** are the predictor variables.

Syntax

The basic syntax for **lm()** function in multiple regression is −

```
lm(y ~ x1+x2+x3...,data)
```

Steps to Perform Multiple Regression in R

1. **Data Collection:** The data to be used in the prediction is collected.
2. **Data Capturing in R:** Capturing the data using the code and importing a CSV file
3. **Checking Data Linearity with R:** It is important to make sure that a linear relationship exists between the dependent and the independent variable. It can be done using scatter plots or the code in R
4. **Applying Multiple Linear Regression in R:** Using code to apply **multiple linear regression in R** to obtain a set of coefficients.
5. **Making Prediction with R:** A predicted value is determined at the end.

Predicting Medical Expenses Using linear regression

Dataset Description

The insurance.csv file includes 1,338 examples of beneficiaries currently enrolled in the insurance plan, with features indicating characteristics of the patient as well as the total medical expenses charged to the plan for the calendar year. The features are:

• age: An integer indicating the age of the primary beneficiary (excluding those above 64 years, as they are generally covered by the government).

• sex: The policy holder's gender: either male or female.

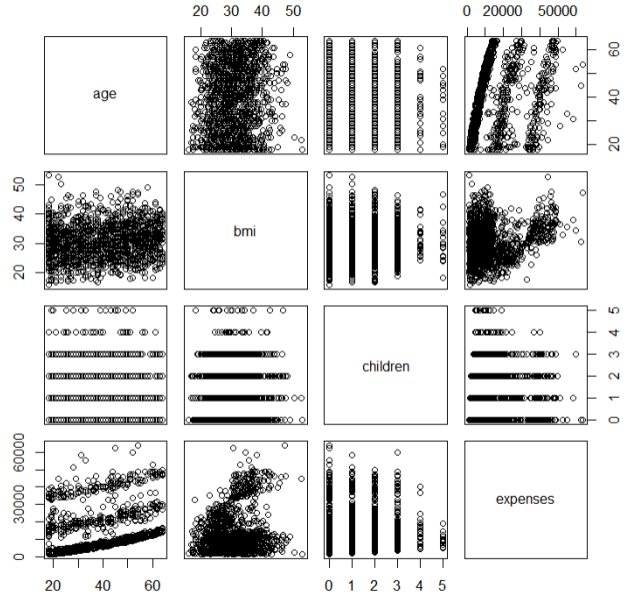
• bmi: The body mass index (BMI), which provides a sense of how over or underweight a person is relative to their height. BMI is equal to weight (in kilograms) divided by height (in meters) squared. An ideal BMI is within the range of 18.5 to 24.9.

• children: An integer indicating the number of children/dependents covered by the insurance plan.

• smoker: A yes or no categorical variable that indicates whether the insured regularly smokes tobacco.

• region: The beneficiary's place of residence in the US, divided into four geographic regions: northeast, southeast, southwest, or northwest.

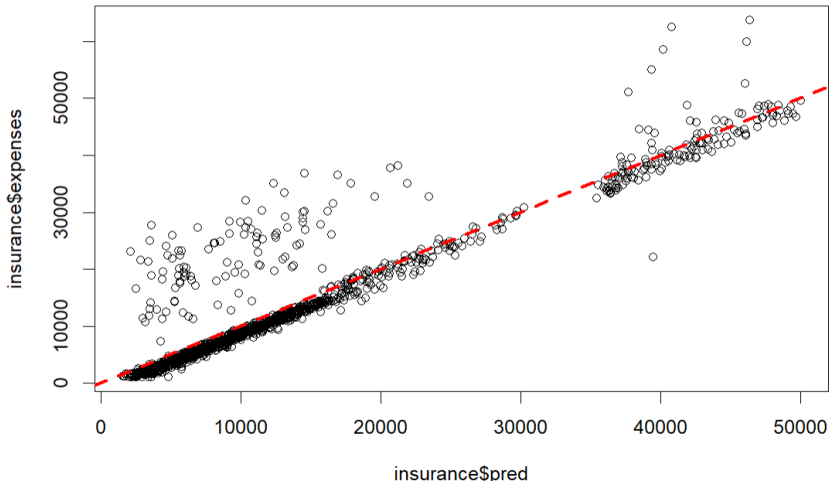
```
> insurance <- read.csv("C:/Users/Dell/Downloads/insurance.csv")

>   View(insurance)

> str(insurance)
'data.frame':   1338 obs. of  7 variables:
 $ age     : int  19 18 28 33 32 31 46 37 37 60 ...
 $ sex     : chr  "female" "male" "male" "male" ...
 $ bmi     : num  27.9 33.8 33 22.7 28.9 25.7 33.4 27.7 29.8 25.8 ...
 $ children: int  0 1 3 0 0 0 1 3 2 0 ...
 $ smoker  : chr  "yes" "no" "no" "no" ...
 $ region  : chr  "southwest" "southeast" "southeast" "northwest" ...
 $ expenses: num  16885 1726 4449 21984 3867 .
..
> summary(insurance$expenses)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1122    4740    9382   13270   16640   63770

> table(insurance$region)

northeast northwest southeast southwest
      324       325       364       325

> cor(insurance[c("age", "bmi", "children", "expenses")])
              age       bmi   children   expenses
age      1.0000000 0.10934101 0.04246900 0.29900819
bmi      0.1093410 1.00000000 0.01264471 0.19857626
children 0.0424690 0.01264471 1.00000000 0.06799823
expenses 0.2990082 0.19857626 0.06799823 1.00000000

> pairs(insurance[c("age", "bmi", "children", "expenses")])
```




```
> install.packages("psych")
package ‘mnormt’ successfully unpacked and MD5 sums checked
```

```
The downloaded binary packages are in
        C:\Users\Dell\AppData\Local\Temp\Rtmpg5QpCU\downloaded_packages
installing the source package ‘psych’
trying URL 'https://cran.rstudio.com/src/contrib/psych_2.3.9.tar.gz'
Content type 'application/x-gzip' length 1640189 bytes (1.6 MB)
downloaded 1.6 MB
DONE (psych)

> ins_model <- lm(expenses ~ ., data = insurance)
> ins_model
Call:
lm(formula = expenses ~ ., data = insurance)
Coefficients:
   (Intercept)           age       sexmale           bmi      children
     -11941.6          256.8         -131.4          339.3          475.7
    smokeryes  regionnorthwest  regionsoutheast  regionsouthwest
      23847.5         -352.8        -1035.6         -959.3
> summary(ins_model)
Call:
lm(formula = expenses ~ ., data = insurance)
Residuals:
    Min      1Q  Median      3Q     Max
-11302.7  -2850.9   -979.6   1383.9  29981.7
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    -11941.6      987.8 -12.089  < 2e-16 ***
age              256.8       11.9  21.586  < 2e-16 ***
sexmale         -131.3      332.9  -0.395 0.693255
bmi              339.3       28.6  11.864  < 2e-16 ***
children         475.7      137.8   3.452 0.000574 ***
smokeryes      23847.5      413.1  57.723  < 2e-16 ***
regionnorthwest   -352.8      476.3  -0.741 0.458976
regionsoutheast  -1035.6      478.7  -2.163 0.030685 *
regionsouthwest   -959.3      477.9  -2.007 0.044921 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6062 on 1329 degrees of freedom
Multiple R-squared:  0.7509,            Adjusted R-squared:  0.7494
F-statistic: 500.9 on 8 and 1329 DF,  p-value: < 2.2e-16

> insurance$age2 <- insurance$age^2

> insurance$bmi30 <- ifelse(insurance$bmi >= 30, 1, 0)

> ins_model2 <- lm(expenses ~ age + age2 + children + bmi + sex + bmi30*smoker +
region, data = insurance)
```

```
> summary(ins_model2)

Call:
lm(formula = expenses ~ age + age2 + children + bmi + sex + bmi30 *
   smoker + region, data = insurance)

Residuals:
    Min      1Q  Median      3Q     Max
-17297.1  -1656.0  -1262.7   -727.8  24161.6

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      139.0053  1363.1359   0.102 0.918792
age             -32.6181    59.8250  -0.545 0.585690
age2              3.7307     0.7463   4.999 6.54e-07 ***
children        678.6017   105.8855   6.409 2.03e-10 ***
bmi             119.7715    34.2796   3.494 0.000492 ***
sexmale        -496.7690   244.3713  -2.033 0.042267 *
bmi30          -997.9355   422.9607  -2.359 0.018449 *
smokeryes     13404.5952   439.9591  30.468  < 2e-16 ***
regionnorthwest  -279.1661   349.2826  -0.799 0.424285
regionsoutheast  -828.0345   351.6484  -2.355 0.018682 *
regionsouthwest -1222.1619   350.5314  -3.487 0.000505 ***
bmi30:smokeryes 19810.1534   604.6769  32.762  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4445 on 1326 degrees of freedom
Multiple R-squared:  0.8664,            Adjusted R-squared:  0.8653
F-statistic: 781.7 on 11 and 1326 DF,  p-value: < 2.2e-16

> insurance$pred <- predict(ins_model2, insurance)

> cor(insurance$pred, insurance$expenses)
[1] 0.9307999


> plot(insurance$pred, insurance$expenses)
> abline(a = 0, b = 1, col = "red", lwd = 3, lty = 2)
```




```
> predict(ins_model2, data.frame(age = 30, age2 = 30^2, children = 2, bmi = 30, sex = "male", bmi30 = 1, smoker = "no", region = "northeast"))
    1
5973.774

> predict(ins_model2, data.frame(age = 30, age2 = 30^2, children = 2, bmi = 30, sex = "female", bmi30 = 1, smoker = "no", region = "northeast"))
    1
6470.543
```

Regression tree

Regression trees are a type of machine learning model used for predictive modeling and regression analysis. They are a variant of decision trees specifically designed for solving regression problems, where the goal is to predict a continuous numeric outcome. In a Regression tree the target variable is continuous and the tree is used to predict its value. For example, if the response variable is the temperature of the day. Regression trees are interpretable and easy to visualize, making them a useful tool for understanding the relationships between input features and the target variable

A regression tree is a hierarchical structure composed of nodes and branches. Each node represents a decision point based on one of the input features. The branches emanating from each node represent the possible outcomes of the decision. The tree-building process begins with the root node, which includes all the data. At each node, the algorithm selects a feature and a splitting criterion that partitions the data into two or more subsets. The goal is to find splits that maximize the reduction in the variability (e.g., minimizing mean squared error) of the target variable within each subset. The process continues recursively until a stopping condition is met. Typically, this stopping condition could be a maximum tree depth, a minimum number of data points in a node, or a predefined level of homogeneity in the target variable within a node. When the stopping condition is met, a node becomes a leaf node, and it contains the predicted value for the target variable.

Regression tree models can be evaluated using various metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), or R-squared (R^2) to measure how well they fit the training data and how well they generalize to new data.

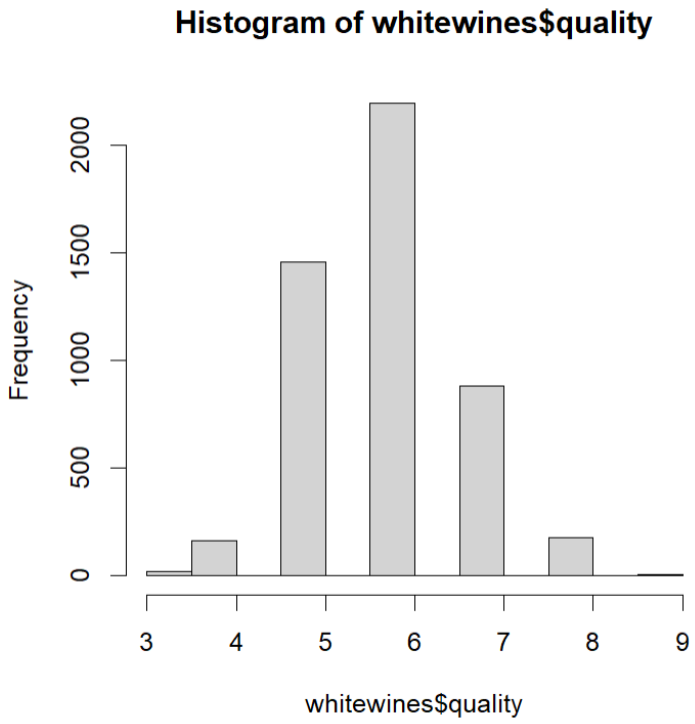
Mean Square Error

A measure that tells us how much our predictions deviate from the original target and that's the entry-point of mean square error.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Estimating the quality of wines with regression trees and model trees

```
>whitewines <- read.csv("C:/Users/Dell/Downloads/whitewines.csv")

>   View(whitewines)

> str(whitewines)
'data.frame':       4898 obs. of  12 variables:
 $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
 $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0.22 ...
 $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0.43 ...
 $ residual.sugar      : num  20.7 1.6 6.9 8.5 8.5 6.9 7 20.7 1.6 1.5 ...
 $ chlorides           : num  0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.045 0.049 0.044 ...
 $ free.sulfur.dioxide : num  45 14 30 47 47 30 30 45 14 28 ...
 $ total.sulfur.dioxide: num  170 132 97 186 186 97 136 170 132 129 ...
 $ density             : num  1.001 0.994 0.995 0.996 0.996 ...
 $ pH                  : num  3 3.3 3.26 3.19 3.19 3.26 3.18 3 3.3 3.22 ...
 $ sulphates           : num  0.45 0.49 0.44 0.4 0.4 0.44 0.47 0.45 0.49 0.45 ...
 $ alcohol             : num  8.8 9.5 10.1 9.9 9.9 10.1 9.6 8.8 9.5 11 ...
 $ quality             : int  6 6 6 6 6 6 6 6 6 6 ...

> hist(whitewines$quality)
```



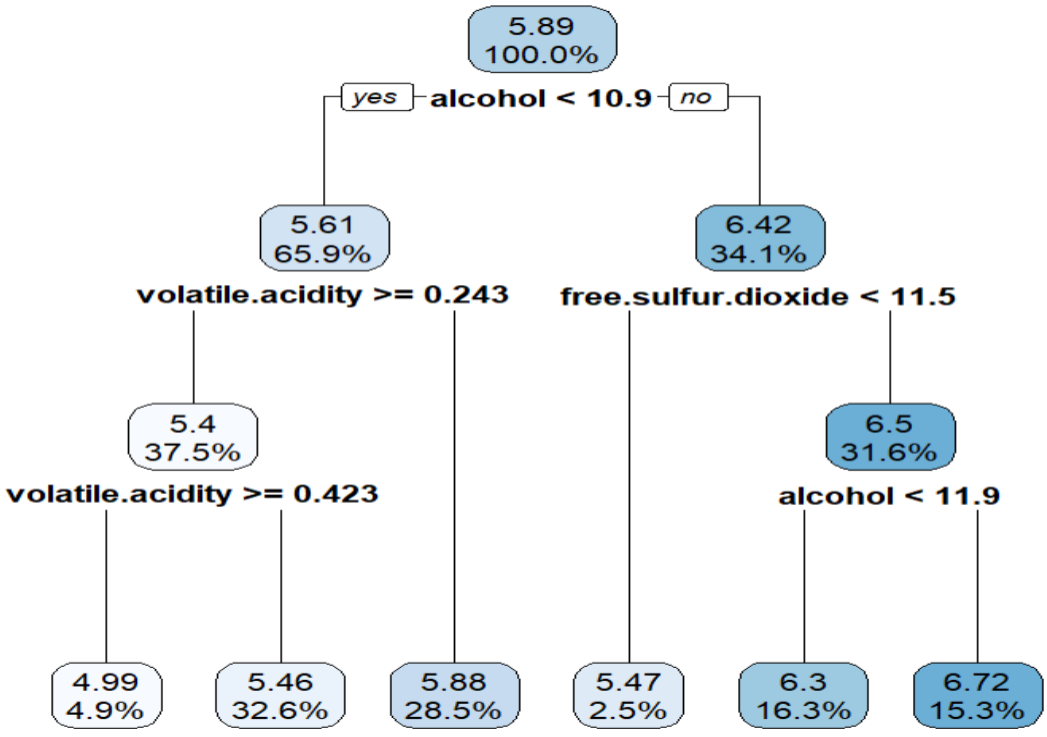

```
> summary(whitewines)
 fixed.acidity    volatile.acidity  citric.acid     residual.s
ugar
 Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0
.600
 1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1
.700
 Median : 6.800   Median :0.2600   Median :0.3200   Median : 5
.200
 Mean   : 6.855   Mean   :0.2782   Mean   :0.3342   Mean   : 6
.391
 3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9
.900
 Max.   :14.200   Max.   :1.1000   Max.   :1.6600   Max.   :65
.800
   chlorides       free.sulfur.dioxide total.sulfur.dioxide
density
 Min.   :0.00900   Min.   :  2.00      Min.   :  9.0        Mi
n.   :0.9871
 1st Qu.:0.03600   1st Qu.: 23.00      1st Qu.:108.0        1s
t Qu.:0.9917
 Median :0.04300   Median : 34.00      Median :134.0        Me
dian :0.9937
 Mean   :0.04577   Mean   : 35.31      Mean   :138.4        Me
an   :0.9940
 3rd Qu.:0.05000   3rd Qu.: 46.00      3rd Qu.:167.0        3r
d Qu.:0.9961
 Max.   :0.34600   Max.   :289.00      Max.   :440.0        Ma
x.   :1.0390
       pH           sulphates         alcohol         quality
 Min.   :2.720   Min.   :0.2200   Min.   : 8.00   Min.   :3.00
0
 1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50   1st Qu.:5.00
0
 Median :3.180   Median :0.4700   Median :10.40   Median :6.00
0
```

```
 Mean   :3.188   Mean   :0.4898   Mean   :10.51   Mean   :5.87
8
 3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40   3rd Qu.:6.00
0
 Max.   :3.820   Max.   :1.0800   Max.   :14.20   Max.   :9.00
0

> wine_train <-whitewines[1:3750, ]

> wine_test <- whitewines[3751:4898, ]

> library(rpart)

> m.rpart <- rpart(quality ~ ., data = wine_train)

> m.rpart
n= 3750

node), split, n, deviance, yval
      * denotes terminal node

 1) root 3750 3140.06000 5.886933
   2) alcohol< 10.85 2473 1510.66200 5.609381
     4) volatile.acidity>=0.2425 1406  740.15080 5.402560
       8) volatile.acidity>=0.4225 182   92.99451 4.994505 *
       9) volatile.acidity< 0.4225 1224  612.34560 5.463235 *
     5) volatile.acidity< 0.2425 1067  631.12090 5.881912 *
   3) alcohol>=10.85 1277 1069.95800 6.424432
     6) free.sulfur.dioxide< 11.5 93   99.18280 5.473118 *
     7) free.sulfur.dioxide>=11.5 1184  879.99920 6.499155
      14) alcohol< 11.85 611  447.38130 6.296236 *
      15) alcohol>=11.85 573  380.63180 6.715532 *

> summary(m.rpart)
Call:
rpart(formula = quality ~ ., data = wine_train)
  n= 3750

          CP nsplit rel error    xerror       xstd
1 0.17816211      0 1.0000000 1.0004933 0.02388279
2 0.04439109      1 0.8218379 0.8230379 0.02238784
3 0.02890893      2 0.7774468 0.7855091 0.02209431
4 0.01655575      3 0.7485379 0.7582846 0.02095733
5 0.01108600      4 0.7319821 0.7462527 0.02046460
6 0.01000000      5 0.7208961 0.7422445 0.02020423

Variable importance
             alcohol              density            chlorides
                  38                   23                   12
    volatile.acidity total.sulfur.dioxide  free.sulfur.dioxide
                  12                    7                    6
           sulphates                   pH       residual.sugar
                   1                    1                    1

Node number 1: 3750 observations,    complexity param=0.178162
1
  mean=5.886933, MSE=0.8373493
  left son=2 (2473 obs) right son=3 (1277 obs)
  Primary splits:
      alcohol              < 10.85    to the left,  improve=0.
17816210, (0 missing)
```

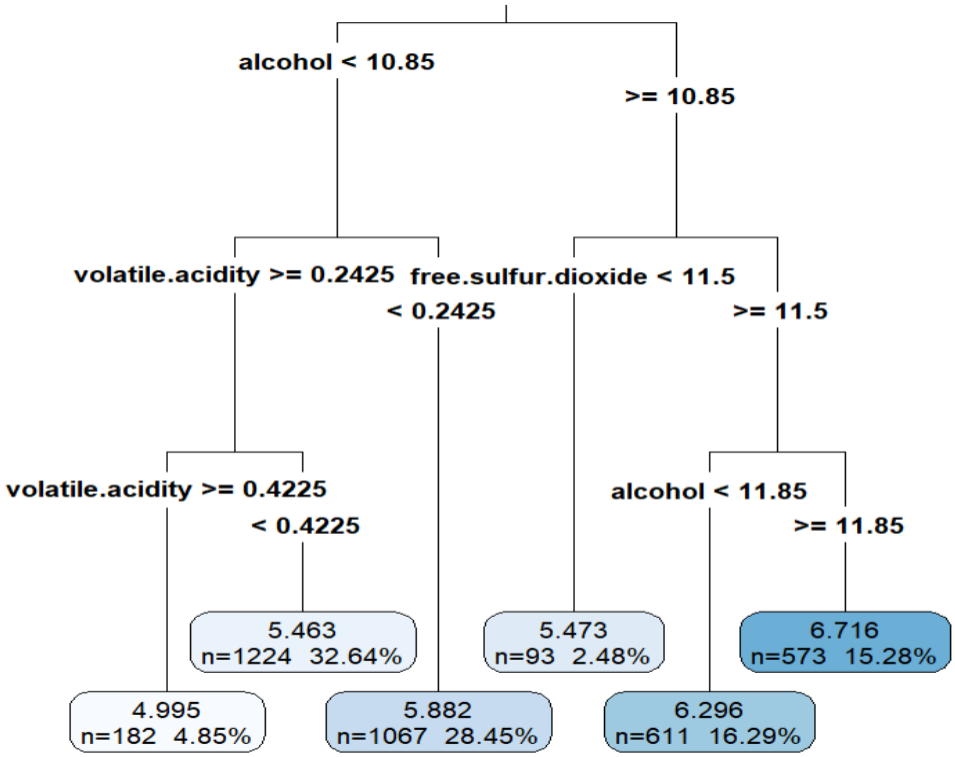
```
      density              < 0.992385 to the right, improve=0.
11980970, (0 missing)
      chlorides            < 0.0395   to the right, improve=0.
08199995, (0 missing)
      total.sulfur.dioxide < 153.5    to the right, improve=0.
03875440, (0 missing)
      free.sulfur.dioxide  < 11.75    to the left,  improve=0.
03632119, (0 missing)
  Surrogate splits:
      density              < 0.99201  to the right, agree=0.86
9, adj=0.614, (0 split)
      chlorides            < 0.0375   to the right, agree=0.77
3, adj=0.334, (0 split)
      total.sulfur.dioxide < 102.5    to the right, agree=0.70
5, adj=0.132, (0 split)
      sulphates            < 0.345    to the right, agree=0.67
0, adj=0.031, (0 split)
      fixed.acidity        < 5.25     to the right, agree=0.66
2, adj=0.009, (0 split)

Node number 2: 2473 observations,    complexity param=0.044391
09
  mean=5.609381, MSE=0.6108623
  left son=4 (1406 obs) right son=5 (1067 obs)
  Primary splits:
      volatile.acidity    < 0.2425   to the right, improve=0.0
9227123, (0 missing)
      free.sulfur.dioxide < 13.5     to the left,  improve=0.0
4177240, (0 missing)
      alcohol             < 10.15    to the left,  improve=0.0
3313802, (0 missing)
      citric.acid         < 0.205    to the left,  improve=0.0
2721200, (0 missing)
      pH                  < 3.325    to the left,  improve=0.0
1860335, (0 missing)
  Surrogate splits:
      total.sulfur.dioxide < 111.5    to the right, agree=0.61
0, adj=0.097, (0 split)
      pH                   < 3.295    to the left,  agree=0.59
8, adj=0.067, (0 split)
      alcohol              < 10.05    to the left,  agree=0.59
0, adj=0.049, (0 split)
      sulphates            < 0.715    to the left,  agree=0.58
4, adj=0.037, (0 split)
      residual.sugar       < 1.85     to the right, agree=0.58
1, adj=0.029, (0 split)

Node number 3: 1277 observations,    complexity param=0.028908
93
  mean=6.424432, MSE=0.8378682
  left son=6 (93 obs) right son=7 (1184 obs)
  Primary splits:
      free.sulfur.dioxide  < 11.5     to the left,  improve=0.
08484051, (0 missing)
      alcohol              < 11.85    to the left,  improve=0.
06149941, (0 missing)
      fixed.acidity        < 7.35     to the right, improve=0.
04259695, (0 missing)
      residual.sugar       < 1.275    to the left,  improve=0.
02795662, (0 missing)
```

```
      total.sulfur.dioxide < 67.5     to the left,  improve=0.
02541719, (0 missing)
  Surrogate splits:
      total.sulfur.dioxide < 48.5     to the left,  agree=0.93
7, adj=0.14, (0 split)

Node number 4: 1406 observations,    complexity param=0.011086
  mean=5.40256, MSE=0.526423
  left son=8 (182 obs) right son=9 (1224 obs)
  Primary splits:
      volatile.acidity     < 0.4225   to the right, improve=0.
04703189, (0 missing)
      free.sulfur.dioxide  < 17.5     to the left,  improve=0.
04607770, (0 missing)
      total.sulfur.dioxide < 86.5     to the left,  improve=0.
02894310, (0 missing)
      alcohol              < 10.25    to the left,  improve=0.
02890077, (0 missing)
      chlorides            < 0.0455   to the right, improve=0.
02096635, (0 missing)
  Surrogate splits:
      density       < 0.99107  to the left,  agree=0.874, adj=
0.027, (0 split)
      citric.acid   < 0.11     to the left,  agree=0.873, adj=
0.022, (0 split)
      fixed.acidity < 9.85     to the right, agree=0.873, adj=
0.016, (0 split)
      chlorides     < 0.206    to the right, agree=0.871, adj=
0.005, (0 split)

Node number 5: 1067 observations
  mean=5.881912, MSE=0.591491

Node number 6: 93 observations
  mean=5.473118, MSE=1.066482

Node number 7: 1184 observations,    complexity param=0.016555
75
  mean=6.499155, MSE=0.7432425
  left son=14 (611 obs) right son=15 (573 obs)
  Primary splits:
      alcohol        < 11.85    to the left,  improve=0.059075
11, (0 missing)
      fixed.acidity  < 7.35     to the right, improve=0.044006
60, (0 missing)
      density        < 0.991395 to the right, improve=0.025224
10, (0 missing)
      residual.sugar < 1.225    to the left,  improve=0.025039
36, (0 missing)
      pH             < 3.245    to the left,  improve=0.024179
36, (0 missing)
  Surrogate splits:
      density              < 0.991115 to the right, agree=0.71
0, adj=0.401, (0 split)
      volatile.acidity     < 0.2675   to the left,  agree=0.66
5, adj=0.307, (0 split)
      chlorides            < 0.0365   to the right, agree=0.63
1, adj=0.237, (0 split)
      total.sulfur.dioxide < 126.5    to the right, agree=0.56
6, adj=0.103, (0 split)
```

```
      residual.sugar       < 1.525    to the left,  agree=0.56
0, adj=0.091, (0 split)

Node number 8: 182 observations
  mean=4.994505, MSE=0.5109588

Node number 9: 1224 observations
  mean=5.463235, MSE=0.5002823

Node number 14: 611 observations
  mean=6.296236, MSE=0.7322117

Node number 15: 573 observations
  mean=6.715532, MSE=0.6642788

> install.packages("rpart.plot")
package 'rpart.plot' successfully unpacked and MD5 sums checke
d

The downloaded binary packages are in
      C:\Users\Dell\AppData\Local\Temp\Rtmp4OHN4U\downloaded_p
ackages

> library(rpart.plot)

> rpart.plot(m.rpart, digits = 3)
```




```
> rpart.plot(m.rpart, digits = 4, fallen.leaves = TRUE, type =
3, extra = 101)
```




```
> p.rpart <- predict(m.rpart, wine_test)

> summary(p.rpart)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.995   5.463   5.882   5.999   6.296   6.716

> summary(wine_test$quality)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   5.000   6.000   5.848   6.000   8.000

> cor(p.rpart, wine_test$quality)
[1] 0.4931608
> MAE <- function(actual, predicted) {
+     mean(abs(actual - predicted))
+ }

> MAE(p.rpart, wine_test$quality)
[1] 0.5732104

> mean(wine_train$quality)
[1] 5.886933

> MAE(5.87, wine_test$quality)
[1] 0.5815679
```

Neural Network

A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain. It is a type of machine learning process, called deep learning, that uses interconnected nodes or neurons in a layered structure that resembles the human brain. It creates an adaptive system that computers use to learn from their mistakes and improve continuously. Thus, artificial neural networks attempt to solve complicated problems, like summarizing documents or recognizing faces, with greater accuracy.

Why are neural networks important?

Neural networks can help computers make intelligent decisions with limited human assistance. This is because they can learn and model the relationships between input and output data that are nonlinear and complex. For instance, they can do the following tasks.

Make generalizations and inferences

Neural networks can comprehend unstructured data and make general observations without explicit training. For instance, they can recognize that two different input sentences have a similar meaning:

- Can you tell me how to make the payment?
- How do I transfer money?

A neural network would know that both sentences mean the same thing. Or it would be able to broadly recognize that Baxter Road is a place, but Baxter Smith is a person's name.

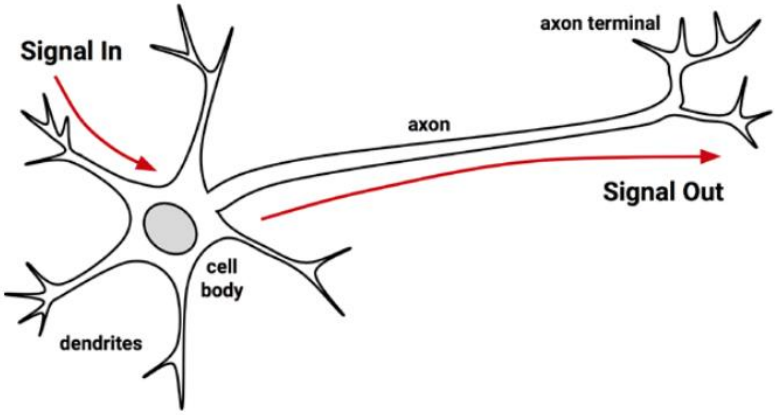
Applications of Neural Networks

Neural networks have several use cases across many industries, such as the following:

- Medical diagnosis by medical image classification
- Targeted marketing by social network filtering and behavioral data analysis
- Financial predictions by processing historical data of financial instruments
- Electrical load and energy demand forecasting
- Process and quality control
- Chemical compound identification

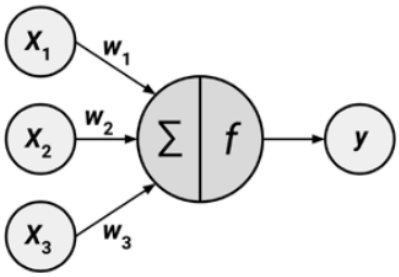
How do neural networks work?

The human brain is the inspiration behind neural network architecture. Human brain cells, called neurons, form a complex, highly interconnected network and send electrical signals to each other to help humans process information. Similarly, an artificial neural network is made of artificial neurons that work together to solve a problem. Artificial neurons are software modules, called nodes, and artificial neural networks are software programs or algorithms that, at their core, use computing systems to solve mathematical calculations.



Incoming signals are received by the cell's dendrites through a biochemical process. The process allows the impulse to be weighted according to its relative importance or frequency. As the cell body begins accumulating the incoming signals, a threshold is reached at which the cell fires and the output signal is transmitted via an electrochemical process down the axon. At the axon's terminals, the electric signal is again processed as a chemical signal to be passed to the neighbouring neurons across a tiny gap known as a synapse.

A directed network diagram defines a relationship between the input signals received by the dendrites (x variables), and the output signal (y variable). Just as with the biological neuron, each dendrite's signal is weighted (w values) according to its importance—ignore, for now, how these weights are determined. The input signals are summed by the cell body and the signal is passed on according to an activation function denoted by f:



A typical artifcial neuron with n input dendrites can be represented by the formula that follows. The w weights allow each of the n inputs (denoted by xi) to contribute a greater or lesser amount to the sum of input signals. The net total is used by the activation function f(x), and the resulting signal, y(x), is the output axon:

$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

Simple neural network architecture

A basic neural network has interconnected artificial neurons in three layers:

Input Layer

Information from the outside world enters the artificial neural network from the input layer. Input nodes process the data, analyze or categorize it, and pass it on to the next layer.

Hidden Layer

Hidden layers take their input from the input layer or other hidden layers. Artificial neural networks can have a large number of hidden layers. Each hidden layer analyzes the output from the previous layer, processes it further, and passes it on to the next layer.

Output Layer

The output layer gives the final result of all the data processing by the artificial neural network. It can have single or multiple nodes. For instance, if we have a binary (yes/no) classification problem, the output layer will have one output node, which will give the result as 1 or 0. However, if we have a multi-class classification problem, the output layer might consist of more than one output node.

Neural networks use neurons defined this way as building blocks to construct complex models of data. Although there are numerous variants of neural networks, each can be defned in terms of the following characteristics:

• An **activation function**, which transforms a neuron's combined input signals into a single output signal to be broadcasted further in the network

• A **network topology** (or architecture), which describes the number of neurons in the model as well as the number of layers and manner in which they are connected

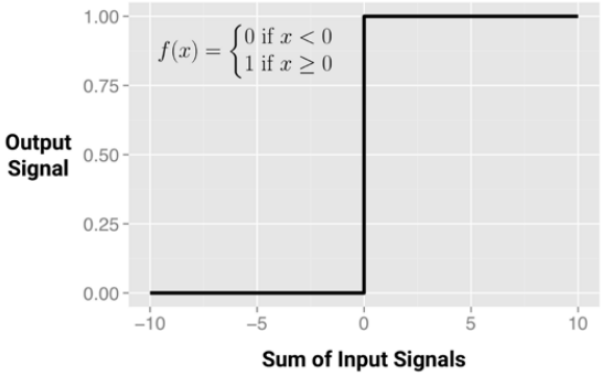
• The **training algorithm** that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal

Activation functions

The activation function is the mechanism by which the artificial neuron processes incoming information and passes it throughout the network. Just as the artificial neuron is modeled after the biological version, so is the activation function modeled after nature's design.

In the biological case, the activation function could be imagined as a process that involves summing the total input signal and determining whether it meets the firing threshold. If so, the neuron passes on the signal; otherwise, it does nothing. In ANN terms, this is known as a **threshold activation function**, as it results in an output signal only once a specified input threshold has been attained.

The following figure depicts a typical threshold function; in this case, the neuron fires when the sum of input signals is at least zero. Because its shape resembles a stair, it is sometimes called a **unit step activation function**.



Although the threshold activation function is interesting due to its parallels with biology, it is rarely used in artificial neural networks. Freed from the limitations of biochemistry, the ANN activation functions can be chosen based on their ability to demonstrate desirable mathematical characteristics and accurately model relationships among data

Network topology

The ability of a neural network to learn is rooted in its topology, or the patterns and structures of interconnected neurons. Although there are countless forms of network architecture, they can be differentiated by three key characteristics:

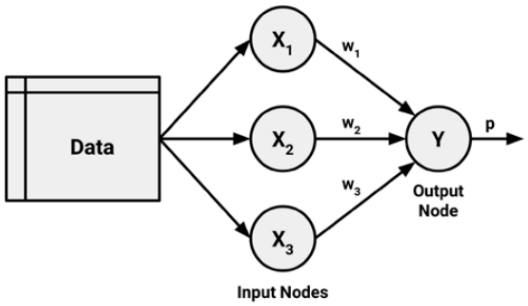
- The number of layers
- Whether information in the network is allowed to travel backward

- The number of nodes within each layer of the network

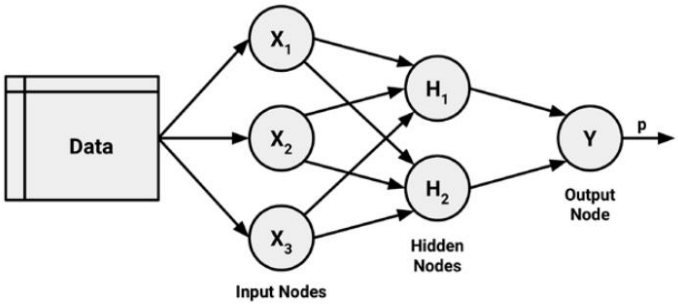
The topology determines the complexity of tasks that can be learned by the network. Generally, larger and more complex networks are capable of identifying more subtle patterns and complex decision boundaries. However, the power of a network is not only a function of the network size, but also the way units are arranged

The number of layers

To defne topology, we need a terminology that distinguishes artificial neurons based on their position in the network. A set of neurons called input nodes receives unprocessed signals directly from the input data. Each input node is responsible for processing a single feature in the dataset; the feature's value will be transformed by the corresponding node's activation function. The signals sent by the input nodes are received by the output node, which uses its own activation function to generate a final prediction (denoted here as p). The input and output nodes are arranged in groups known as layers. Because the input nodes process the incoming data exactly as it is received, the network has only one set of connection weights (labelled here as w1, w2, and w3). It is therefore termed a single-layer network. Single-layer networks can be used for basic pattern classification, particularly for patterns that are linearly separable, but more sophisticated networks are required for most learning tasks



A multilayer network adds one or more hidden layers that process the signals from the input nodes prior to it reaching the output node. Most multilayer networks are fully connected, which means that every node in one layer is connected to every node in the next layer, but this is not required

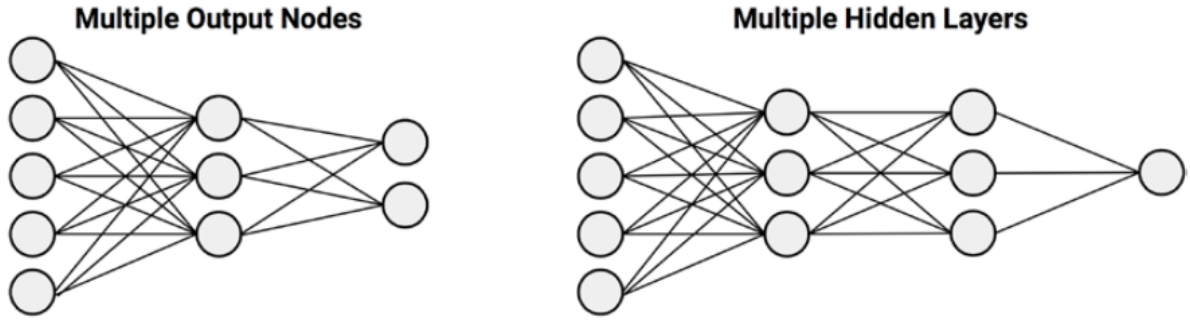


The direction of information travel

Networks in which the input signal is fed continuously in one direction from connection to connection until it reaches the output layer are called **feedforward networks.**

In spite of the restriction on information flow, feedforward networks offer a surprising amount of flexibility. For instance, the number of levels and nodes at each level can be varied, multiple outcomes can be modelled simultaneously, or multiple hidden layers can be applied. A neural network with multiple hidden layers is called a Deep Neural Network

(DNN) and the practice of training such network is sometimes referred to as deep learning.



In contrast, a recurrent network (or feedback network) allows signals to travel in both directions using loops. This property, which more closely mirrors how a biological neural network works, allows extremely complex patterns to be learned. The addition of a short-term memory, or delay, increases the power of recurrent networks immensely.

Backpropagation is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization. Backpropagation in neural network is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network.

Support Vector Machine

A support vector machine (SVM) is a supervised learning algorithm used in machine learning. SVMs are used for: Classification, Regression, Outlier detection**.** It is a supervised machine learning problem where we try to find a hyperplane that best separates the two classes.

Types of Support Vector Machine Algorithms

1. Linear SVM

When the data is **perfectly linearly separable** only then we can use Linear SVM. Perfectly linearly separable means that the data points can be classified into 2 classes by using a single straight line(if 2D).

2. Non-Linear SVM

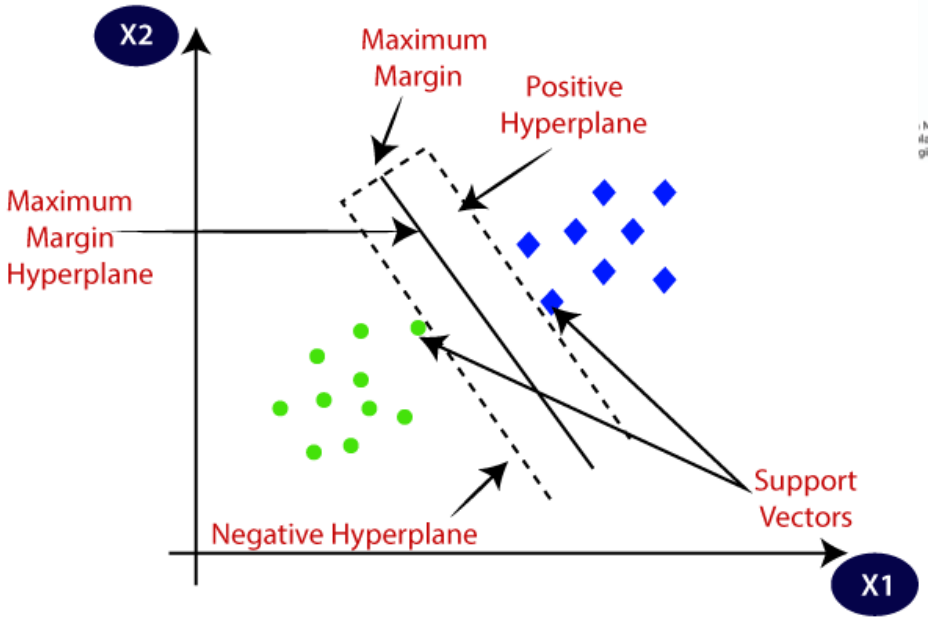
When the data is **not linearly separable** then we can use Non-Linear SVM, which means when the data points cannot be separated into 2 classes by using a straight line (if 2D) then we use some advanced techniques like kernel tricks to classify them. In most real-world applications we do not find linearly separable datapoints hence we use kernel trick to solve them.

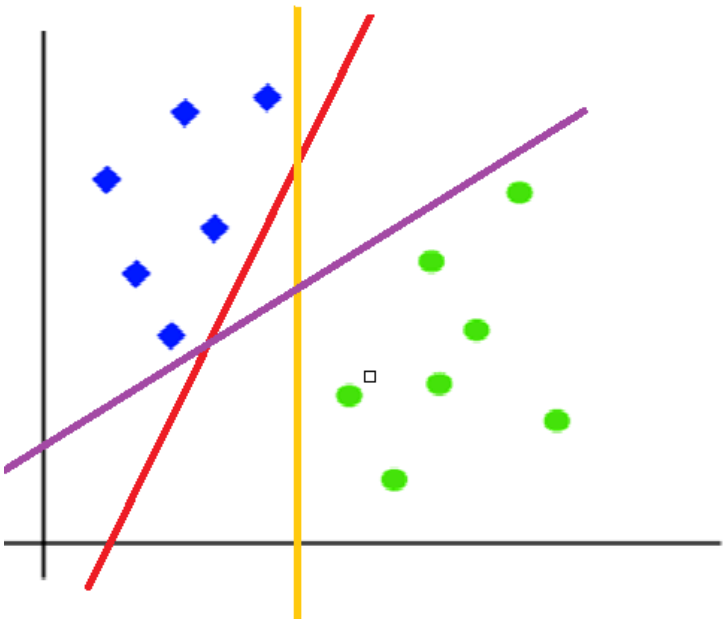
Important Terms

- **Support Vectors:** These are the points that are closest to the hyperplane. A separating line will be defined with the help of these data points.
- **Margin:** it is the distance between the hyperplane and the observations closest to the hyperplane (support vectors). In SVM large margin is considered a good margin. There are two types of margins **hard margin** and **soft margin.**

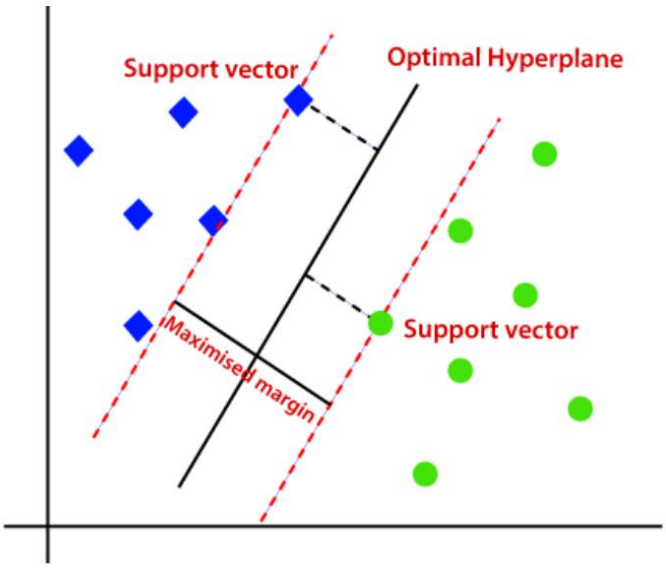
- **Hard Margin:** The maximum-margin hyperplane or the hard margin hyperplane is a hyperplane that properly separates the data points of different categories without any misclassifications.

- **Soft Margin:** When the data is not perfectly separable or contains outliers, SVM permits a soft margin technique. Each data point has a slack variable introduced by the soft-margin SVM formulation, which softens the strict margin requirement and permits certain misclassifications or violations. It discovers a compromise between increasing the margin and reducing violations

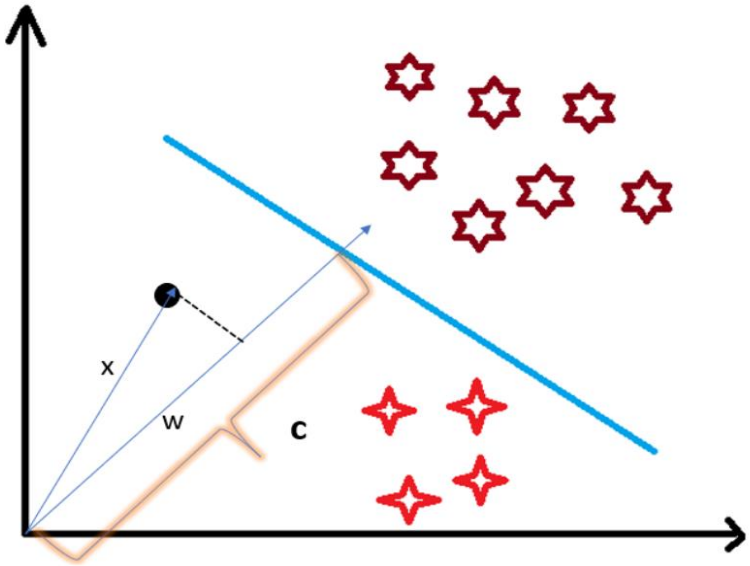




The **best hyperplane is that plane that has the maximum distance** from both the classes, and this is the main aim of SVM. This is done by finding different hyperplanes which classify the labels in the best way then it will choose the one which is farthest from the data points or the one which has a maximum margin.



Consider a random point X and we want to know whether it lies on the right side of the plane or the left side of the plane (positive or negative). To find this first we assume this point is a vector (X) and then we make a vector (w) which is perpendicular to the hyperplane. Let's say the distance of vector w from origin to decision boundary is 'c'. Now we take the projection of X vector on w.



Understanding Dot-Product

- The dot product is a mathematical operation used to find the projection of one vector onto another vector.
- It is defined as the product of the magnitudes of the vectors and the cosine of the angle between them.

$$\vec{X}.\vec{w} = c \quad (the\ point\ lies\ on\ the\ decision\ boundary)$$

$$\vec{X}.\vec{w} > c\ (positive\ samples)$$

$$\vec{X}.\vec{w} < c\ (negative\ samples)$$

1. If the dot product is greater than 'c,' the point lies on the right side of the hyperplane.
2. If the dot product is less than 'c,' the point is on the left side.

3. If the dot product is equal to 'c,' the point lies on the decision boundary.

The formula for the dot product of two vectors, X and w, is given by:

Dot Product (X · w) = |X| * |w| * cos(θ)

where:

- |X| is the magnitude of vector X.
- |w| is the magnitude of vector w.
- θ is the angle between vectors X and w.

Vector geometry defines the distance between these two planes as

$$\frac{2}{||\vec{w}||}$$

Applications

- Facial recognitions
- Classification of microarray gene expression data in the field of bioinformatics to identify cancer or other genetic diseases
- Text categorization such as identification of the language used in a document
- The detection events like combustion engine failure, security breaches, or earthquakes

Performing OCR with SVMs

Support vector machine syntax

using the `ksvm()` function in the `kernlab` package

Building the model:

```
m <- ksvm(target ~ predictors, data = mydata,
          kernel = "rbfdot", C = 1)
```

- `target` is the outcome in the `mydata` data frame to be modeled
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` specifies the data frame in which the `target` and `predictors` variables can be found
- `kernel` specifies a nonlinear mapping such as `"rbfdot"` (radial basis), `"polydot"` (polynomial), `"tanhdot"` (hyperbolic tangent sigmoid), or `"vanilladot"` (linear)
- C is a number that specifies the cost of violating the constraints, i.e., how big of a penalty there is for the "soft margin." Larger values will result in narrower margins

The function will return a SVM object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "response")
```

- `m` is a model trained by the `ksvm()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier
- `type` specifies whether the predictions should be `"response"` (the predicted class) or `"probabilities"` (the predicted probability, one column per class level).

The function will return a vector (or matrix) of predicted classes (or probabilities) depending on the value of the type parameter.

Example:

```
letter_classifier <- ksvm(letter ~ ., data =
  letters_train, kernel = "vanilladot")
letter_prediction <- predict(letter_classifier,
  letters_test)
```

Required packages

```
library(kernlab)
```

Step 1 - Collecting data

Step 2 - Exploring and preparing the data

```
letters <- read.csv("letterdata.csv", stringsAsFactors = TRUE)
#Exploring the structure of the data frame
str(letters)
```

```
## 'data.frame':    20000 obs. of  17 variables:
##  $ letter: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10 13 ...
##  $ xbox  : int  2 5 4 7 2 4 4 1 2 11 ...
##  $ ybox  : int  8 12 11 11 1 11 2 1 2 15 ...
##  $ width : int  3 3 6 6 3 5 5 3 4 13 ...
##  $ height: int  5 7 8 6 1 8 4 2 4 9 ...
##  $ onpix : int  1 2 6 3 1 3 4 1 2 7 ...
##  $ xbar  : int  8 10 10 5 8 8 8 8 10 13 ...
##  $ ybar  : int  13 5 6 9 6 8 7 2 6 2 ...
##  $ x2bar : int  0 5 2 4 6 6 6 2 2 6 ...
##  $ y2bar : int  6 4 6 6 6 9 6 2 6 2 ...
##  $ xybar : int  6 13 10 4 6 5 7 8 12 12 ...
##  $ x2ybar: int  10 3 3 4 5 6 6 2 4 1 ...
##  $ xy2bar: int  8 9 7 10 9 6 6 8 8 9 ...
##  $ xedge : int  0 2 3 6 1 0 2 1 1 8 ...
##  $ xedgey: int  8 8 7 10 7 8 8 6 6 1 ...
##  $ yedge : int  0 4 3 2 5 9 7 2 1 1 ...
##  $ yedgex: int  8 10 9 8 10 7 10 7 7 8 ...
```

We will split the data set to train and test, the data set is already randomized

```
letters_train <- letters[1:16000, ]
letters_test <- letters[16001:20000, ]
```

Step 3 - training a model on the data

```
#We will apply a linear kernel vanilla on the training data set
```

```
letter_classifier <- ksvm(letter~., data = letters_train,
                          kernel = "vanilladot")
```

```
##  Setting default kernel parameters
```

```
letter_classifier
```

```
## Support Vector Machine object of class "ksvm"

## SV type: C-svc  (classification)

##  parameter : cost C = 1

## Linear (vanilla) kernel function.

## Number of Support Vectors : 7037

## Objective Function Value : -14.1746 -20.0072 -23.5628 -6.2009 -7.5524 -32.7694 -4
9.9786 -18.1824 -62.1111 -32.7284 -16.2209 -32.2837 -28.9777 -51.2195 -13.276 -35.6
217 -30.8612 -16.5256 -14.6811 -32.7475 -30.3219 -7.7956 -11.8138 -32.3463 -13.126
2 -9.2692 -153.1654 -52.9678 -76.7744 -119.2067 -165.4437 -54.6237 -41.9809 -67.26
88 -25.1959 -27.6371 -26.4102 -35.5583 -41.2597 -122.164 -187.9178 -222.0856 -21.4
765 -10.3752 -56.3684 -12.2277 -49.4899 -9.3372 -19.2092 -11.1776 -100.2186 -29.13
97 -238.0516 -77.1985 -8.3339 -4.5308 -139.8534 -80.8854 -20.3642 -13.0245 -82.515
1 -14.5032 -26.7509 -18.5713 -23.9511 -27.3034 -53.2731 -11.4773 -5.12 -13.9504 -4.4
982 -3.5755 -8.4914 -40.9716 -49.8182 -190.0269 -43.8594 -44.8667 -45.2596 -13.556
1 -17.7664 -87.4105 -107.1056 -37.0245 -30.7133 -112.3218 -32.9619 -27.2971 -35.58
36 -17.8586 -5.1391 -43.4094 -7.7843 -16.6785 -58.5103 -159.9936 -49.0782 -37.8426
-32.8002 -74.5249 -133.3423 -11.1638 -5.3575 -12.438 -30.9907 -141.6924 -54.2953 -1
79.0114 -99.8896 -10.288 -15.1553 -3.7815 -67.6123 -7.696 -88.9304 -47.6448 -94.371
8 -70.2733 -71.5057 -21.7854 -12.7657 -7.4383 -23.502 -13.1055 -239.9708 -30.4193 -
25.2113 -136.2795 -140.9565 -9.8122 -34.4584 -6.3039 -60.8421 -66.5793 -27.2816 -2
14.3225 -34.7796 -16.7631 -135.7821 -160.6279 -45.2949 -25.1023 -144.9059 -82.235
2 -327.7154 -142.0613 -158.8821 -32.2181 -32.8887 -52.9641 -25.4937 -47.9936 -6.89
91 -9.7293 -36.436 -70.3907 -187.7611 -46.9371 -89.8103 -143.4213 -624.3645 -119.2
204 -145.4435 -327.7748 -33.3255 -64.0607 -145.4831 -116.5903 -36.2977 -66.3762 -
44.8248 -7.5088 -217.9246 -12.9699 -30.504 -2.0369 -6.126 -14.4448 -21.6337 -57.308
4 -20.6915 -184.3625 -20.1052 -4.1484 -4.5344 -0.828 -121.4411 -7.9486 -58.5604 -21.
4878 -13.5476 -5.646 -15.629 -28.9576 -20.5959 -76.7111 -27.0119 -94.7101 -15.1713
-10.0222 -7.6394 -1.5784 -87.6952 -6.2239 -99.3711 -101.0906 -45.6639 -24.0725 -61.
7702 -24.1583 -52.2368 -234.3264 -39.9749 -48.8556 -34.1464 -20.9664 -11.4525 -12
3.0277 -6.4903 -5.1865 -8.8016 -9.4618 -21.7742 -24.2361 -123.3984 -31.4404 -88.390
1 -30.0924 -13.8198 -9.2701 -3.0823 -87.9624 -6.3845 -13.968 -65.0702 -105.523 -13.7
403 -13.7625 -50.4223 -2.933 -8.4289 -80.3381 -36.4147 -112.7485 -4.1711 -7.8989 -1.
2676 -90.8037 -21.4919 -7.2235 -47.9557 -3.383 -20.433 -64.6138 -45.5781 -56.1309 -
6.1345 -18.6307 -2.374 -72.2553 -111.1885 -106.7664 -23.1323 -19.3765 -54.9819 -34.
2953 -64.4756 -20.4115 -6.689 -4.378 -59.141 -34.2468 -58.1509 -33.8665 -10.6902 -5
3.1387 -13.7478 -20.1987 -55.0923 -3.8058 -60.0382 -235.4841 -12.6837 -11.7407 -17.
3058 -9.7167 -65.8498 -17.1051 -42.8131 -53.1054 -25.0437 -15.302 -44.0749 -16.958
2 -62.9773 -5.204 -5.2963 -86.1704 -3.7209 -6.3445 -1.1264 -122.5771 -23.9041 -355.0
145 -31.1013 -32.619 -4.9664 -84.1048 -134.5957 -72.8371 -23.9002 -35.3077 -11.711
9 -22.2889 -1.8598 -59.2174 -8.8994 -150.742 -1.8533 -1.9711 -9.9676 -0.5207 -26.922
9 -30.429 -5.6289

## Training error : 0.130062
```

Step 4 - Evaluating Model Performance

```
letter_predictions <- predict(letter_classifier, letters_test)
head(letter_predictions)
```

```
## [1] U N V X N H
## Levels: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

To examine how well our classifier performed, we need to compare the predicted letter to the true letter in the testing dataset

```
table(letter_predictions, letters_test$letter)
```

```
##
## letter_predictions   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
##                  A 144   0   0   0   0   0   0   0   0   1   0   0   1   2   2
##                  B   0 121   0   5   2   0   1   2   0   0   1   0   1   0   0
##                  C   0   0 120   0   4   0  10   2   2   0   1   3   0   0   2
##                  D   2   2   0 156   0   1   3  10   4   3   4   3   0   5   5
##                  E   0   0   5   0 127   3   1   1   0   0   3   4   0   0   0
##                  F   0   0   0   0   0 138   2   2   6   0   0   0   0   0   0
##                  G   1   1   2   1   9   2 123   2   0   0   1   2   1   0   1
##                  H   0   0   0   1   0   1   0 102   0   2   3   2   3   4  20
##                  I   0   1   0   0   0   1   0   0 141   8   0   0   0   0   0
##                  J   0   1   0   0   0   1   0   2   5 128   0   0   0   0   1
##                  K   1   1   9   0   0   0   2   5   0   0 118   0   0   2   0
##                  L   0   0   0   0   2   0   1   1   0   0   0 133   0   0   0
##                  M   0   0   1   1   0   0   1   1   0   0   0   0 135   4   0
##                  N   0   0   0   0   0   1   0   1   0   0   0   0   0 145   0
##                  O   1   0   2   1   0   0   1   2   0   1   0   0   0   1  99
##                  P   0   0   0   1   0   2   1   0   0   0   0   0   0   0   2
##                  Q   0   0   0   0   0   0   0   8   2   0   0   0   3   0   0   3
##                  R   0   7   0   0   1   0   3   8   0   0  13   0   0   1   1
##                  S   1   1   0   0   1   0   3   0   1   1   0   1   0   0   0
##                  T   0   0   0   0   3   2   0   0   0   0   1   0   0   0   0
##                  U   1   0   3   1   0   0   0   2   0   0   0   0   0   0   1
##                  V   0   0   0   0   0   1   3   4   0   0   0   0   1   2   1
##                  W   0   0   0   0   0   0   1   0   0   0   0   0   2   0   0
```

```
##           X  0  1  0  0  2  0  0  1  3  0  1  6  0  0  1
##           Y  3  0  0  0  0  0  0  1  0  0  0  0  0  0  0
##           Z  2  0  0  0  1  0  0  0  3  4  0  0  0  0  0
##
## letter_predictions   P   Q   R   S   T   U   V   W   X   Y   Z
##           A  0  5  0  1  1  1  0  1  0  0  1
##           B  2  2  3  5  0  0  2  0  1  0  0
##           C  0  0  0  0  0  0  0  0  0  0  0
##           D  3  1  4  0  0  0  0  0  3  3  1
##           E  0  2  0 10  0  0  0  0  2  0  3
##           F 16  0  0  3  0  0  1  0  1  2  0
##           G  2  8  2  4  3  0  0  0  1  0  0
##           H  0  2  3  0  3  0  2  0  0  1  0
##           I  1  0  0  3  0  0  0  0  5  1  1
##           J  1  3  0  2  0  0  0  0  1  0  6
##           K  1  0  7  0  1  3  0  0  5  0  0
##           L  0  1  0  5  0  0  0  0  0  0  1
##           M  0  0  0  0  0  3  0  8  0  0  0
##           N  0  0  3  0  0  1  0  2  0  0  0
##           O  3  3  0  0  0  3  0  0  0  0  0
##           P 130  0  0  0  0  0  0  0  0  1  0
##           Q  1 124  0  5  0  0  0  0  0  2  0
##           R  1  0 138  0  1  0  1  0  0  0  0
##           S  0 14  0 101  3  0  0  0  2  0 10
##           T  0  0  0  3 133  1  0  0  0  2  2
##           U  0  0  0  0  0 152  0  0  1  1  0
##           V  0  3  1  0  0  0 126  1  0  4  0
##           W  0  0  0  0  0  4  4 127  0  0  0
##           X  0  0  0  1  0  0  0  0 137  1  1
##           Y  7  0  0  0  3  0  0  0  0 127  0
##           Z  0  0  0 18  3  0  0  0  0  0 132
```

To summarize the TRUE and False matching we will create a vector for this job

```
agreement <- letter_predictions == letters_test$letter
```

```
table(agreement)
## agreement
## FALSE  TRUE
##   643  3357
prop.table(table(agreement))
## agreement
##   FALSE    TRUE
## 0.16075 0.83925
```

We can see the accuracy of the model is around 84 percent.

Step 5 - Improving Model Performance

Changing the SVM kernel function to Gaussian RBF kernel.

```
#Building the model
letter_classifier_rbf <- ksvm(letter~., data = letters_train,
                              kernel = "rbfdot")
#create predictions
letter_predictions_rbf <- predict(letter_classifier_rbf, letters_test)
#extract accuracy
agreement_rbf <- letter_predictions_rbf == letters_test$letter
table(agreement_rbf)
## agreement_rbf
## FALSE  TRUE
##   277  3723
prop.table(table(agreement_rbf))
## agreement_rbf
##   FALSE    TRUE
## 0.06925 0.93075
```