Machine Learning

Release 01

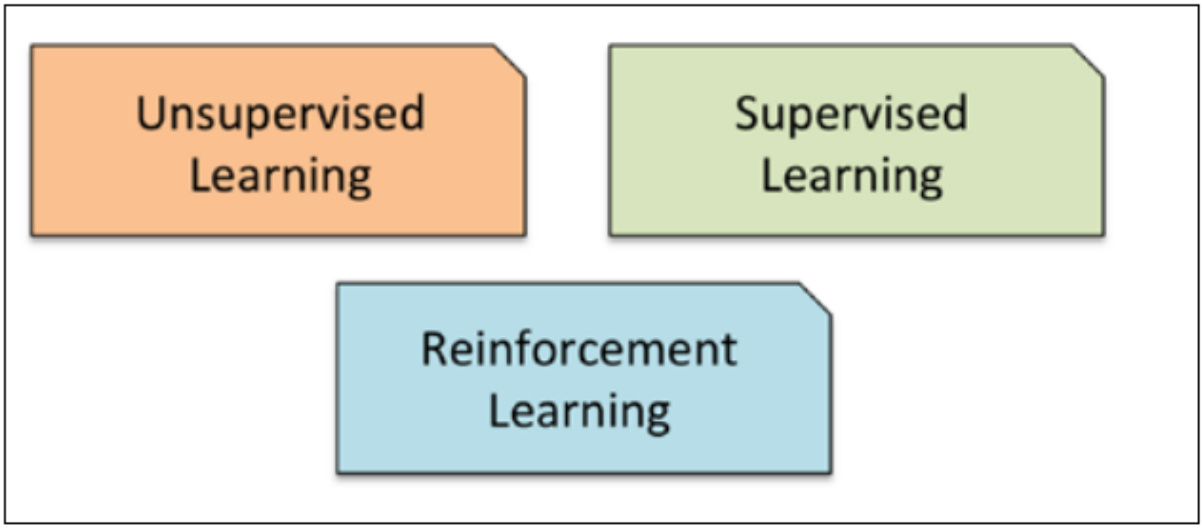
Yavar T. Azar

Aug 22, 2022

Contents

1 Introduction

1.1 Types of ML

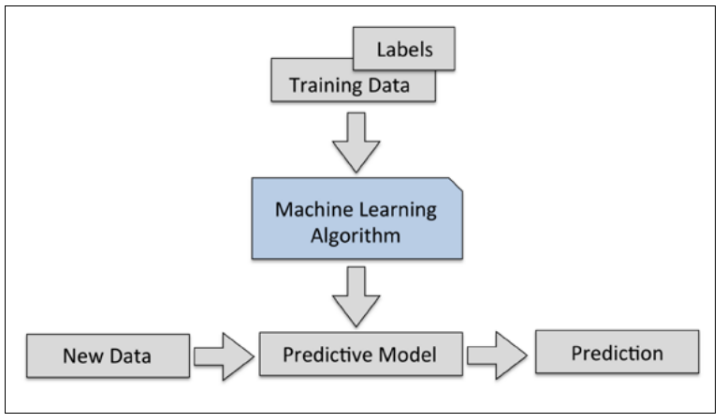


1.2 Supervised Learning

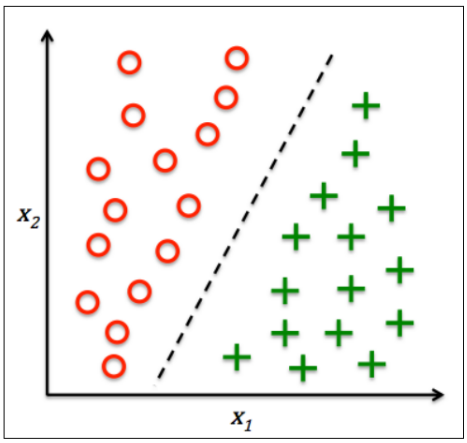
Here, the term supervised refers to a set of samples where the desired output signals (labels) are already known.

For example email with *spam* or *non-spam* labels and predicting the class of new email is a sample of **classification**

Another subcategory of supervised learning is **regression**, where the outcome signal is a continuous value

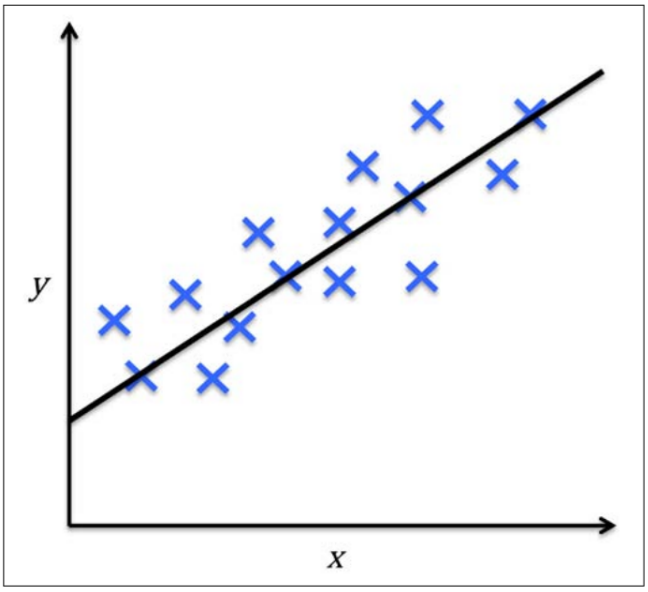


A sample two-dimensional data binary classification



1.2.1 Regression (continuous outcomes)

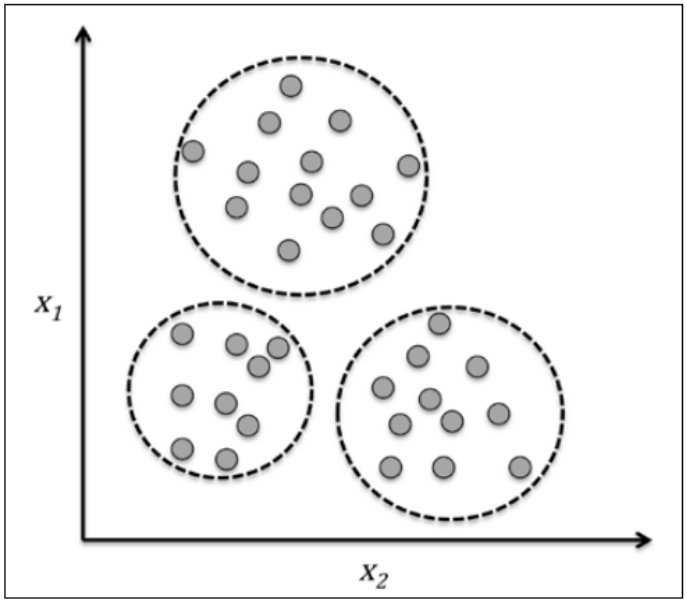
In regression analysis, we are given a number of **predictor (explanatory)** variables and a continuous **response variable (outcome)**, and we try to find a relationship between those variables that allows us to predict an outcome.



1.3 Reinforcement learning

1.4 Unsupervised learning

Simple 2D clustering example based on the similarity of their features x1 and x2

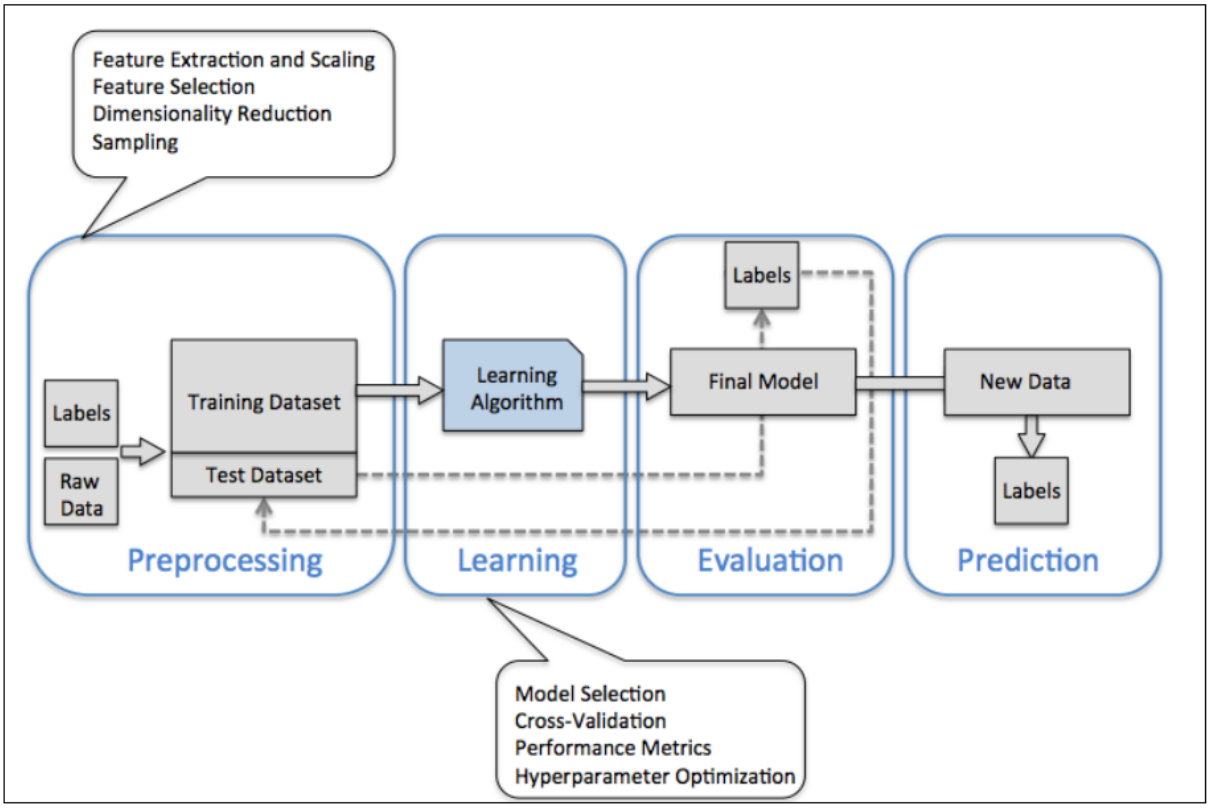


1.5 Dimensionality reduction

$x_j^{(i)}$ we will use the superscript **(i)** to refer to the ith training sample, and the subscript **j** to refer to the jth dimension of the training dataset.

1.6 ML Roadmap

Here



- Preprocessing – getting data into shape

preprocessing ; transforming the features in the range [0,1] or normal distribution

for coorelated and redundant data –> dimensionality reduction techniques

- Training and selecting a predictive model

One legitimate question to ask is: how do we know which model performs well on the final test data-set and real-world data if we don't use this test set for the model selection but keep it for the final model evaluation?

inally, we also cannot expect that the default parameters of the different learning algorithms provided by software libraries are optimal for our specific problem task. hyperparameter optimization techniques

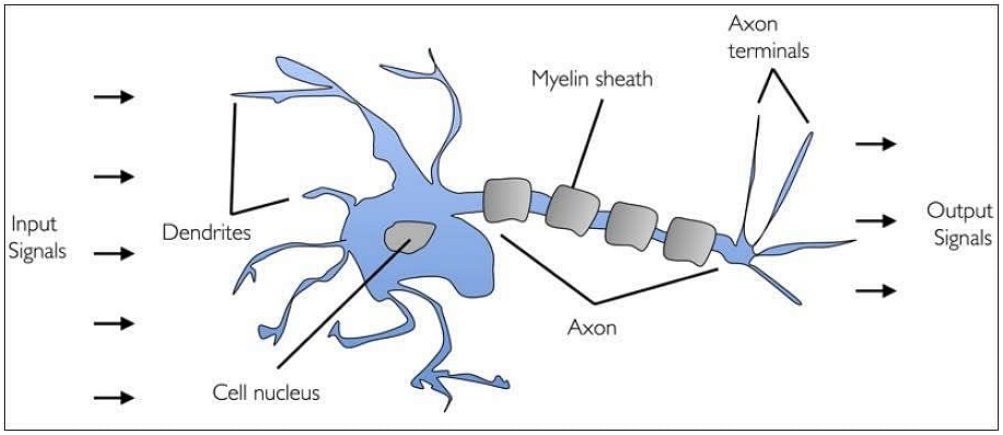
- Evaluating models and predicting unseen data instances

2 Perceptron

2.1 Basic cocepts

The Perceptron is a linear machine learning algorithm for binary classification tasks.

It may be considered one of the first and one of the simplest types of artificial neural networks. It is definitely not “deep” learning but is an important building block.



2.1.1 Perceptron Algorithm

The Perceptron algorithm is a two-class (binary) classification machine learning algorithm.

It is a type of neural network model, perhaps the simplest type of neural network model.

It consists of a single node or neuron that takes a row of data as input and predicts a class label. This is achieved by calculating the weighted sum of the inputs and a bias (set to 1). The weighted sum of the input of the model is called the activation.

- Activation = Weights * Inputs + Bias

If the activation is above **threshold**, the model will output 1.0; otherwise, it will output 0.0.

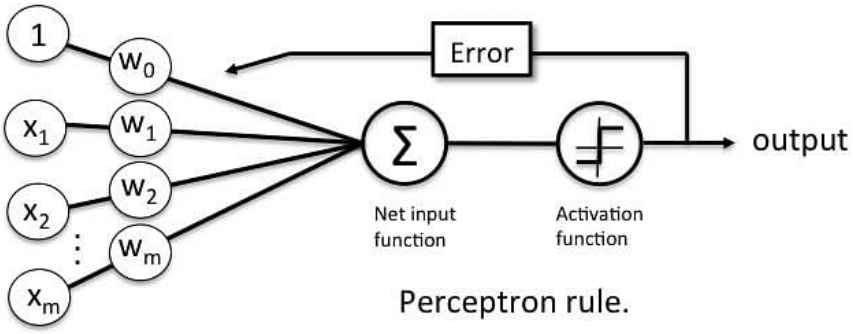
The Perceptron is a linear classification algorithm. This means that it learns a decision boundary that separates two classes using a line (called a **hyperplane**) in the feature space. As

such, it is appropriate for those problems where the classes can be separated well by a line or linear model, referred to as linearly separable.

Examples from the training dataset are shown to the model one at a time, the model makes a prediction, and error is calculated. The weights of the model are then updated to reduce the errors for the example. This is called the Perceptron update rule. This process is repeated for all examples in the training dataset, called an epoch. This process of updating the model using examples is then repeated for many epochs.

Model weights are updated with a small proportion of the error each batch, and the proportion is controlled by a hyperparameter called the learning rate, typically set to a small value. This is to ensure learning does not occur too quickly, resulting in a possibly lower skill model, referred to as premature convergence of the optimization (search) procedure for the model weights.

*weights(t + 1) = weights(t) + learning_rate * (expected_i – predicted_) * input_i* Training is stopped when the error made by the model falls to a low level or no longer improves, or a maximum number of epochs is performed.



Note: The learning rate and number of training epochs are hyperparameters of the algorithm that can be set using heuristics or hyperparameter tuning.

Hands-on

How to Implement the Perceptron Algorithm From Scratch in Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 20 14:52:27 2022

@author: yavar
"""

import numpy as np

class Perceptron(object):
    """Perceptron classifier.
    Parameters
    ------------
    eta : float
    Learning rate (between 0.0 and 1.0)
```

(continues on next page)

(continued from previous page)

```
    n_iter : int
    Passes over the training dataset.
    Attributes
    -----------
    w_ : 1d-array
    Weights after fitting.
    errors_ : list
    Number of misclassifications in every epoch.
    """
    def __init__(self, eta=0.01, n_iter=10):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):


        """Fit training data.
        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
        Training vectors, where n_samples
        is the number of samples and
        n_features is the number of features.
        y : array-like, shape = [n_samples]
        Target values.
        Returns
        -------
        self : object
        """
        self.w_ = np.zeros(1 + X.shape[1])
        self.errors_ = []
        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) + self.w_[0]
    def predict(self, X):
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

After creation of above class we can use it

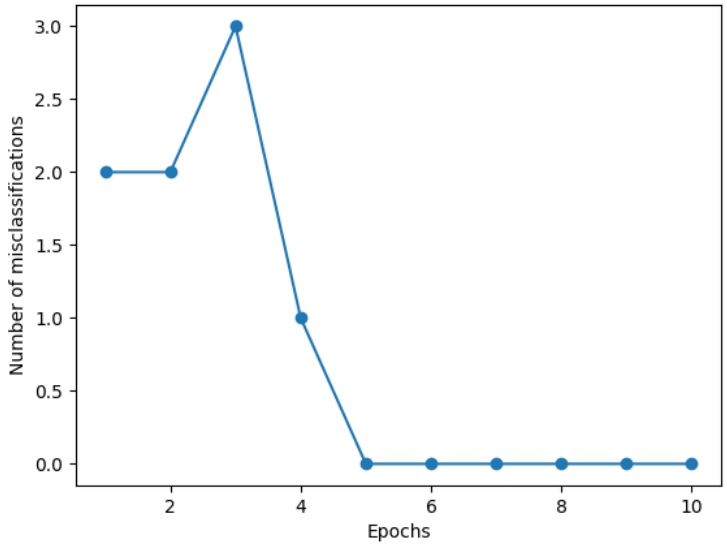
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 20 14:52:27 2022

@author: yavar
```

(continues on next page)

(continued from previous page)

```
"""

from percept import *
import matplotlib.pyplot as plt
import numpy as np


import pandas as pd


df = pd.read_csv("iris.data")


y = df.iloc[0:100, 4].values

y = np.where(y == 'Iris-setosa', -1, 1)


X = df.iloc[0:100, [0, 2]].values



# plt.scatter(X[:50, 0], X[:50, 1], color='red', marker='o', label='setosa')

# plt.scatter(X[50:100, 0], X[50:100, 1],color='blue', marker='x', label='versicolor')
# plt.xlabel('petal length')
# plt.ylabel('sepal length')
# plt.legend(loc='upper left')
#plt.show()


ppn = Perceptron(eta=0.1, n_iter=10)


jj=ppn.fit(X, y)

plt.plot(range(1, len(ppn.errors_) + 1), ppn.errors_,marker='o')


plt.xlabel('Epochs')


plt.ylabel('Number of misclassifications')


plt.show()
```

here is coode run result



2.2 Scikit-learn perceptron

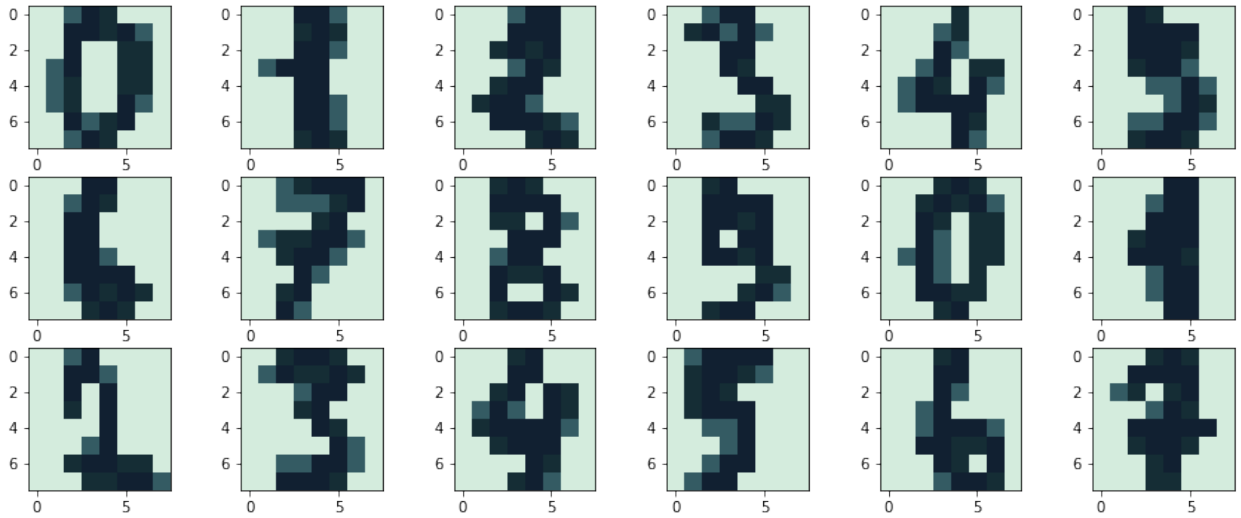
Here we will try perceptron inside scikit-learn library

```
import matplotlib.pyplot as plt

import numpy as np
from matplotlib import colors
ll = ['#112031', '#152D35','#345B63', '#D4ECDD']
ll.reverse()
cmap = colors.ListedColormap(ll)

from sklearn.datasets import load_digits
from sklearn.linear_model import Perceptron
X, y = load_digits(return_X_y=True)
#what is random stat?
clf = Perceptron(tol=1e-1, random_state=15, l1_ratio=0.25)
clf.fit(X, y)
```

let's take a look to the first 18 data inside X dataset

```
test = np.zeros((18,8,8))

for i in range(18):
    test[i,:,:]=np.reshape(X[i],(8, 8))
```

```
fig, axs = plt.subplots(3,6, figsize=(15, 6))

axs = axs.ravel()


for i in range(18):
    axs[i].imshow(test[i], cmap=cmap)
```



lets see what predicted by our perceptron

```
results= clf.predict(X)

print("   prediction |  real value  |  difference")
print("_____________________________________________\n")

for i in range(18):
    print("{:10d}     | {:10d}    |  {:10d}".format(results[i], y[i], results[i]-y[i]))
```

```
   prediction |  real value  |  difference
_____________________________________________

         0    |           0   |           0
         1    |           1   |           0
         2    |           2   |           0
         3    |           3   |           0
         4    |           4   |           0
         9    |           5   |           4
         6    |           6   |           0
         7    |           7   |           0
         8    |           8   |           0
         9    |           9   |           0
         0    |           0   |           0
         1    |           1   |           0
         2    |           2   |           0
         3    |           3   |           0
         4    |           4   |           0
         5    |           5   |           0
         6    |           6   |           0
         7    |           7   |           0
```

lets see how many results are accurate

```
diff = results-y

nonzeros = diff[diff!=0]

nzlen=len(nonzeros)
```

(continues on next page)

(continued from previous page)

```
alldata=len(y)


accuracy = (alldata-nzlen)/alldata

print(accuracy)
```

```
0.9710628825820813
```

Check output of perceprton score method

```
clf.score(X,y)
```

```
0.9710628825820813
```

Our estiamtion is very good !

3

Classification

4

Regression Analysis in ML

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as temperature, age, salary, price, etc.

In Regression, we plot a graph between the variables which best fits the given datapoints, using this plot, the machine learning model can make predictions about the data. In simple words, "Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum." The distance between datapoints and line tells whether a model has captured a strong relationship or not.

Some examples of regression can be as:

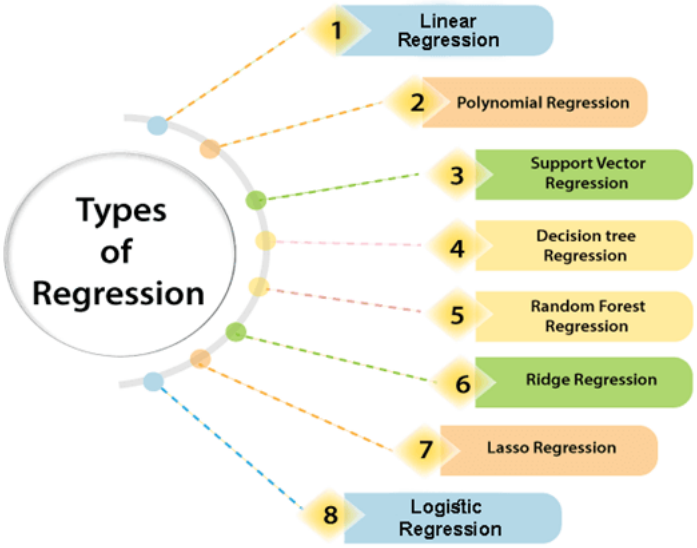
- Prediction of rain using temperature and other factors
- Determining Market trends
- Prediction of road accidents due to rash driving.

4.1 Basic terminology

- Dependent Variable: The main factor in Regression analysis which we want to predict or understand is called the dependent variable. It is also called target variable.
- Independent Variable: The factors which affect the dependent variables or which are used to predict the values of the dependent variables are called independent variable, also called as a predictor.
- Outliers: Outlier is an observation which contains either very low value or very high value in comparison to other observed values. An outlier may hamper the result, so it should be avoided.
- Multicollinearity: If the independent variables are highly correlated with each other than other variables, then such condition is called Multicollinearity. It should not be present in the dataset, because it creates problem while ranking the most affecting variable.

- Underfitting and Overfitting: If our algorithm works well with the training dataset but not well with test dataset, then such problem is called Overfitting. And if our algorithm does not perform well even with training dataset, then such problem is called underfitting.

4.2 Types of regression

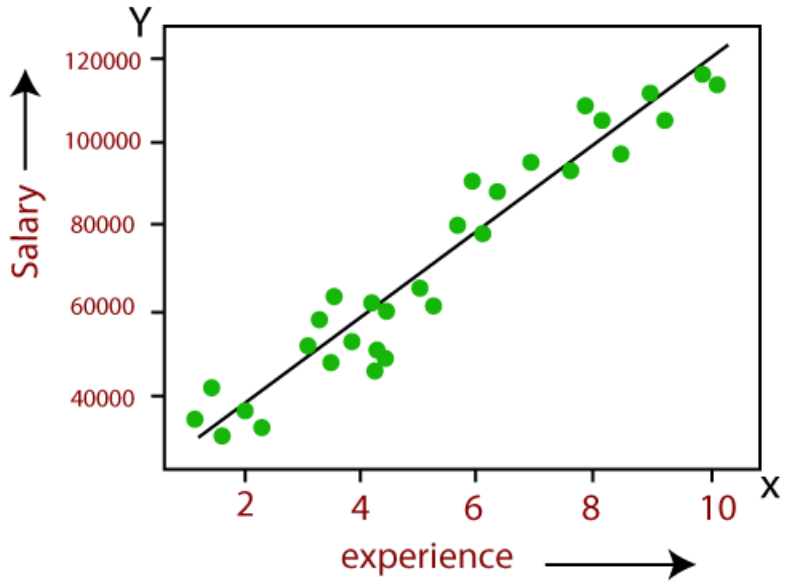


- Linear Regression
- Logistic Regression
- Polynomial Regression
- Support Vector Regression
- Decision Tree Regression
- Random Forest Regression
- Ridge Regression
- Lasso Regression:

Each method can be used for different scenario

4.2.1 Linear regression

Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression.



a mathematical euqation for regression can bes shown

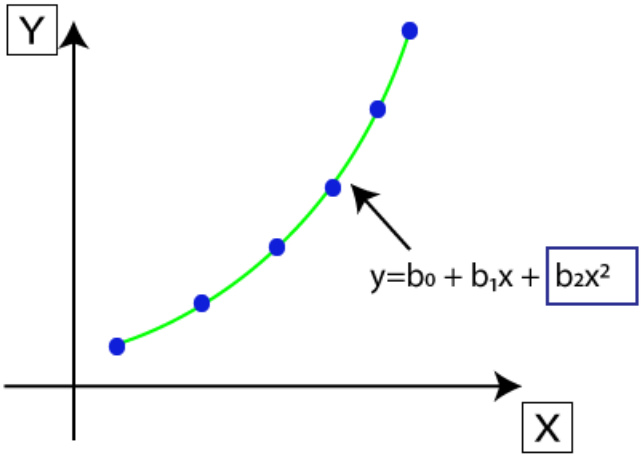
$$Y = aX + b$$

Here, Y = dependent variables (target variables),
X= Independent variables (predictor variables),
a and b are the linear coefficients

4.2.2 Polynomial Regression

Polynomial Regression is a type of regression which models the non-linear dataset using a linear model. It is similar to multiple linear regression, but it fits a non-linear curve between the value of x and corresponding conditional values of y.

Note: In Polynomial regression, the original features are transformed into polynomial features of given degree and then modeled using a linear model. Which means the datapoints are best fitted using a polynomial line.



4.2.3 Logistic Regression

- Logistic regression is another supervised learning algorithm which is used to solve the classification problems. In classification problems, we have dependent variables in a binary or discrete format such as 0 or 1.
- Logistic regression algorithm works with the categorical variable such as 0 or 1, Yes or No, True or False, Spam or not spam, etc.
- Logistic regression uses sigmoid function or logistic function which is a complex cost function. This sigmoid function is used to model the data in logistic regression. The function can be represented as

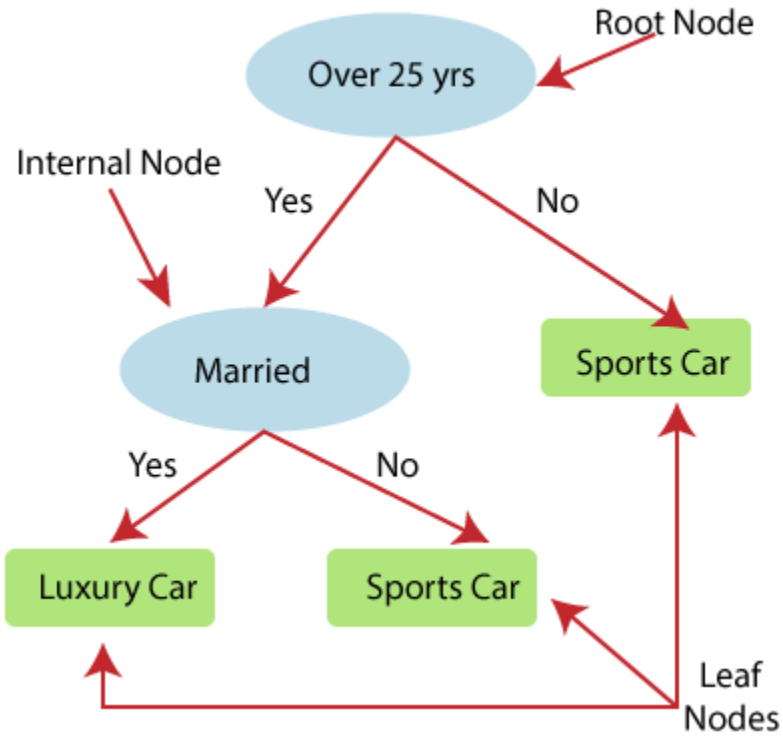
$$f(x) = \frac{1}{1 + e^{-x}}$$

where, f(x)= Output between the 0 and 1 value and x= input to the function

4.2.4 Support Vector Regression

4.2.5 Decision Tree Regression

Decision Tree is a supervised learning algorithm which can be used for solving both classification and regression problems.



5

Data Preprocessing

6

Neural Network

7

Indices and tables

- genindex
- modindex
- search