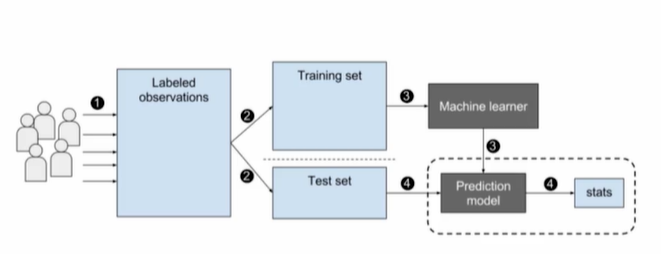
Areas de ML

* Problemas de redes neurales
* Problemas de categorizacion (CLuser)
* Prediccion (REGRESION)
* Randon forest classifiers



Observaciones etiquetadas

Set de entrenamiento

Set de pruebas para validar

El modelo de tomar los datos entrenamiento a Machine learner

Y luego se pasa a produccion

Se tendra un modelo de regression para predecir datos, segun la regression.

Regresion: proyeccion

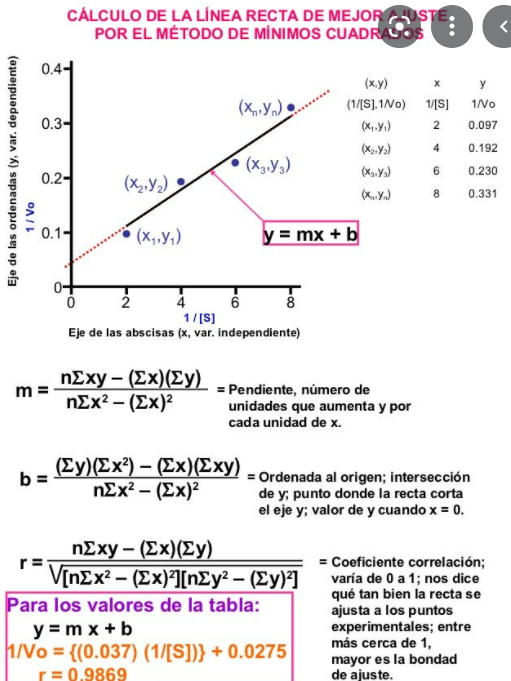
Clasificacion: agrupacion, cluster, prob. De estar en el grupo

ML Supervisado - Algoritmo

|  |  |  |
| --- | --- | --- |
| Algoritmo | Regresion | Clasificacion |
| Naïve Bayes |  | X |
| K-nearest neighbors | X | x |
| Decision tree | X | X |
| Random Forest | X | x |

y = ax + b

Metodo de minimos cuadrados



import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

    # number of observations/points

    n = np.size(x)

    # mean of x and y vector

    m\_x = np.mean(x)

    m\_y = np.mean(y)

    # calculating cross-deviation and deviation about x

    SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x

    SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x

    # calculating regression coefficients

    b\_1 = SS\_xy / SS\_xx

    b\_0 = m\_y - b\_1\*m\_x

    return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

    # plotting the actual points as scatter plot

    plt.scatter(x, y, color = "m",

               marker = "o", s = 30)

    # predicted response vector

    y\_pred = b[0] + b[1]\*x

    # plotting the regression line

    plt.plot(x, y\_pred, color = "g")

    # putting labels

    plt.xlabel('x')

    plt.ylabel('y')

    # function to show plot

    plt.show()

def main():

    # observations / data

    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients

    b = estimate\_coef(x, y)

    print("Estimated coefficients:\nb\_0 = {}  \

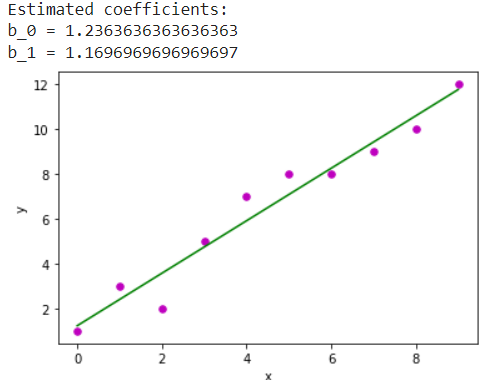
          \nb\_1 = {}".format(b[0], b[1]))

    # plotting regression line

    plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

    main()



import numpy as np

from sklearn.linear\_model import LinearRegression

x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))

y = np.array([5, 20, 14, 32, 22, 38])

model = LinearRegression().fit(x, y)

r\_sq = model.score(x, y)

print('intercept:', model.intercept\_)

print('slope:', model.coef\_)

print('coefficient of determination:', r\_sq)

intercept: 5.633333333333329

slope: [0.54]

coefficient of determination: 0.7158756137479542

model = LinearRegression()

Options

- fit\_intercept is a Boolean (True by default) that decides whether to calculate the intercept 𝑏₀ (True) or consider it equal to zero (False).

- normalize is a Boolean (False by default) that decides whether to normalize the input variables (True) or not (False).

- copy\_X is a Boolean (True by default) that decides whether to copy (True) or overwrite the input variables (False).

- n\_jobs is an integer or None (default) and represents the number of jobs used in parallel computation. None usually means one job and -1 to use all processors.