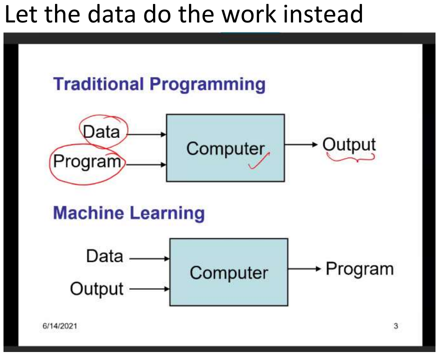
RAJA REDDY PUNDRA-197167-NOTES SUMMER TRAINING-ML-2021

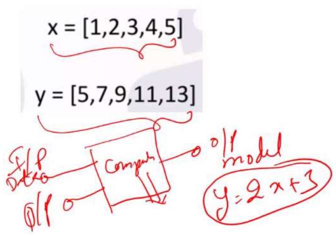
Supervidsed algorithm

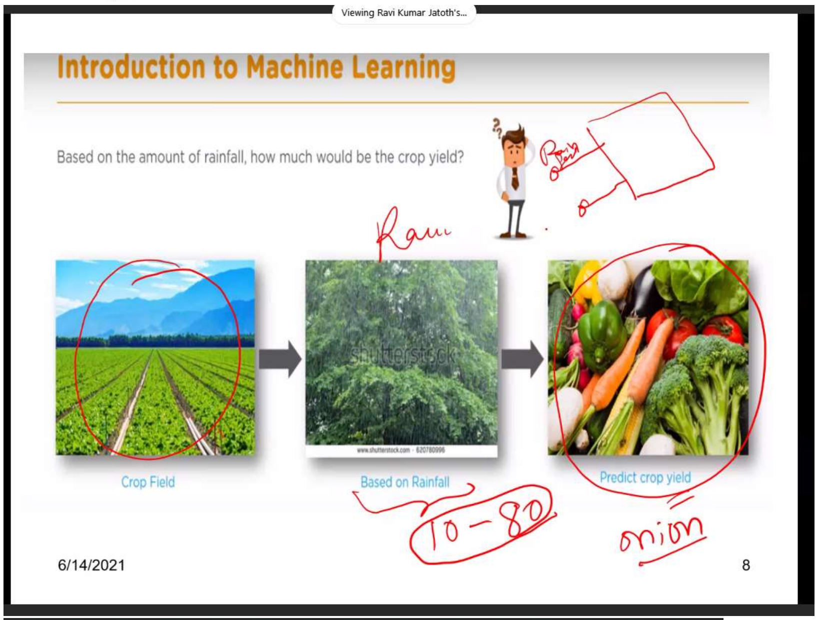
- Basic linear regression
  - Least square method
  - Gradient descent method
- Multiple linear regression
  - Image processing
- Un supervised
  - K means clustered
    - Swiggy, realestate,making them into groups and assigning nearest members;
    - Image processing,image compression;
  - K nearest member
- NEURAL NETWORK
  - Back propa
  - CNN
  - RNN
- Image processing
- GANS

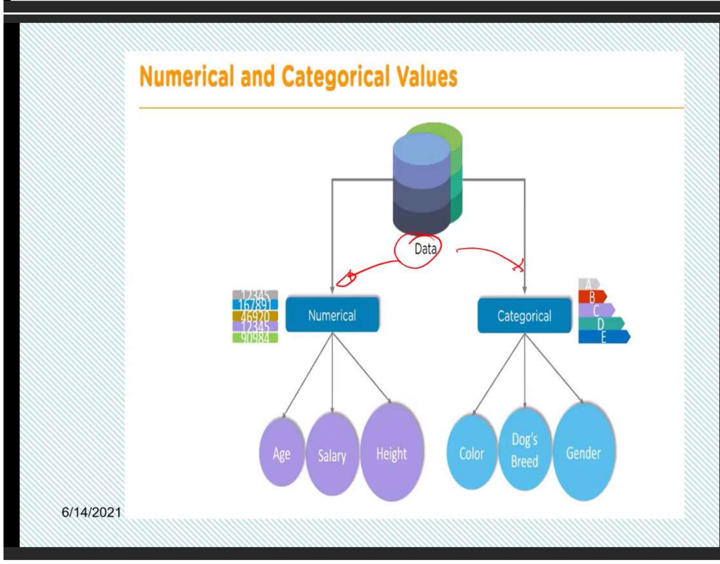
14-06-2021

- Introduction to machine learning and linear regression
  - Automating Autmation;
  - Getting computers to program themselves
  - Writing software is the bottleneck
  - Let the data do the work instead
  - 

- What is ML
  - Branch of AI



- 

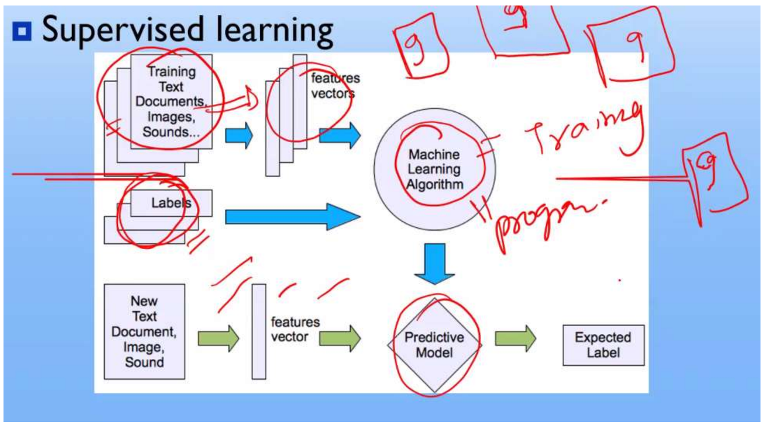
- 

-

❖ Machine learning algo

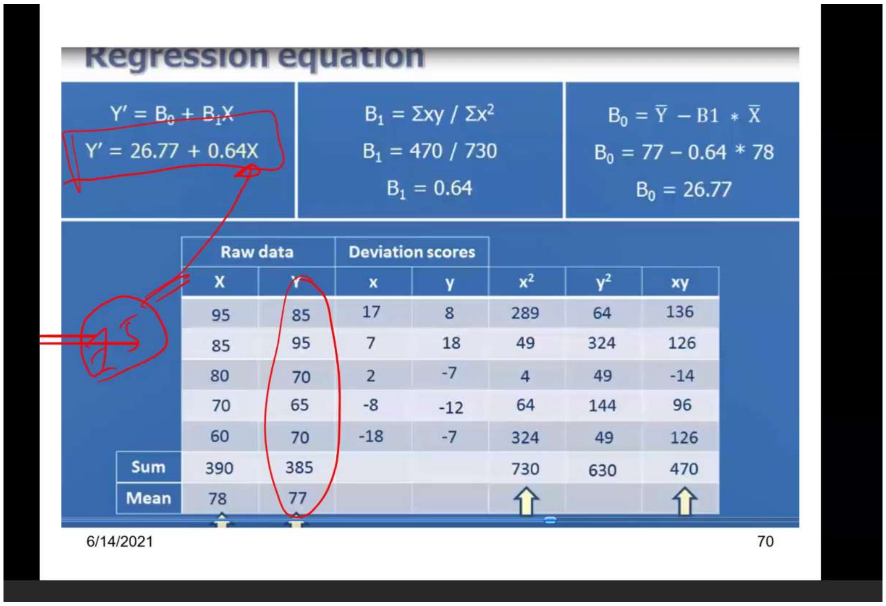
- Supervised

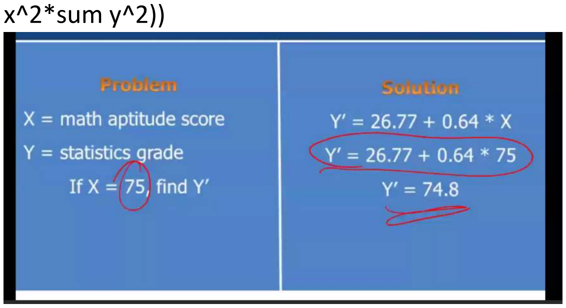
- Input
  - Featured vectors
    - Training text
    - Documents
    - Images
    - Sounds
    - Etc
  - Labels

❖ 

Classification	Regression
Two or more labels Ex: spam or not spam, Obese or not obese,etc Methods:logistic regression	Continuous quantity Ex: weather prediction COST PREDICTION Grade prediction Methods:linear regression

- Regression(estimation of relation between the variables)
  - Real-estate cost, runs based on previous performance, etc.
  - We use least square for checking;



- Coefficient of determination-R^2
  - Ranges from 0-1
  - Near to 0 don't use regression
  - Else use regression
  - C of Determination is related to C of correlation(r_xy= sum xy / sqrt(sum x^2*sum y^2))
  - 

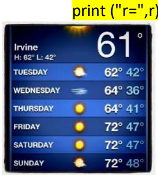
  - Avoid extrapolation ☹ (only between the min and max of our input to be predicted)

Coding part:

- Basic code

```
import numpy as np
import matplotlib.pyplot as plt

x=np.array([95,85,80,70,60])
y=np.array([85,95,70,65,70])
#x is independent and y is dependent variable,
# we are storing those data in the array

n=np.size(x);
m_x,m_y=np.mean(x),np.mean(y)#mean
```

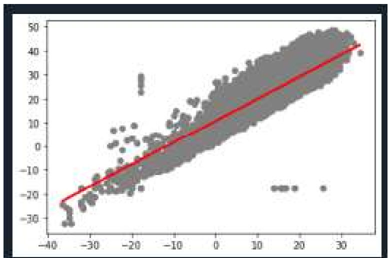
```
ss_xy=np.sum(x*y)-n*(m_x*m_y)#
ss_xx=np.sum(x*x)-n*m_x*m_x

b0_1=ss_xy/ss_xx# slope
b0_0=m_y-b0_1*m_x#intercept

np.append(x,70)#to predict the value of 70,
np.append(y,b0_0+b0_1*70)#predicted value
y_pred=b0_0+b0_1*x#list of predicted values
print("intercept:",b0_0)
print("slope:    ",b0_1)
plt.scatter(x,y)#plotting the actual values on graph
plt.plot(x,y_pred,color='r',marker='o')#plotting the predicted values on graph
from sklearn.metrics import r2_score
r2=r2_score(y,y_pred)#greater the score(coefficient of determination) better the prediction

print("r2 =",r2)
r=r2**0.5
print ("r=",r)
```

- 

- Weather prediction

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
dataset=pd.read_csv("C:/Users/rajar/Documents/.summercoding/ML/to train/wearther/weather.csv")dataset.plot(x='MinTemp',y='MaxTemp',style='o')
plt.title('min temp vs max temp')
plt.xlabel('MinTemp')
plt.ylabel('MaxTemp')
plt.plot()
plt.show()

X=dataset['MinTemp'].values.reshape(-1,1)
y=dataset['MaxTemp'].values.reshape(-1,1)

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)

regressor=LinearRegression()
regressor.fit(X_train,y_train)

print(regressor.intercept_)
print(regressor.coef_)
y_pred=regressor.predict(X_test)
df=pd.DataFrame({'Actual':y_test.flatten(),'predicted ':y_pred.flatten()})
print(df);
plt.scatter(X_test,y_test,color='gray')
plt.plot(X_test,y_pred,color='red',linewidth=2)
plt.show()

from sklearn.metrics import r2_score
r2=r2_score(y_test,y_pred)
```

```
print("r2 score ",r2)
print("the regression eqation : y=10.6619*x+0.92")
```

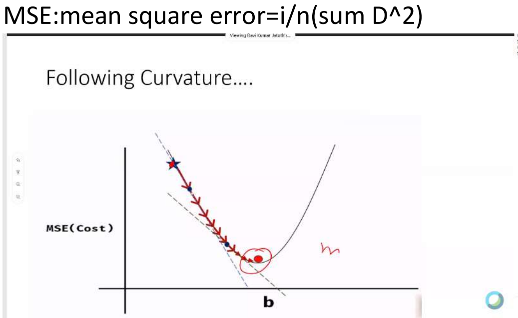


```
LR.py )
[10.66185201]
[[0.92033997]]
          Actual  predicted
0      28.888889   33.670351
1      31.111111   30.091251
2      27.222222   26.512151
3      28.888889   31.113851
4      23.333333   15.774852
...          ...         ...
23803  32.777778   32.136451
23804  32.222222   29.068651
23805  31.111111   32.647751
23806  31.111111   30.602551
23807  36.666667   31.625151

[23808 rows x 2 columns]
r2 score  0.7670218843587764
the regression eqation : y=10.6619*x+0.92
```

- Corona data is having very random values,
- We cannot apply this linear regression when the data is not linear;
- We can apply this to predict our grades even we calculate the grades using BELL CURVE, we can predict efficiently with previous data;

14-06-2021

- Graient Descent method: best fit for given data based on MSE
  - MSE:mean square error=i/n(sum D^2)



  -
  -

- The problem arises when ti is dependent on 2 variales,
  - We take partial derivatives of that surface

Example: find the partial derivatives of **f(x,y,z) = x⁴ − 3xyz** using "curly dee" notation

$f(x,y,z) = x^4 - 3xyz$

$$\frac{\partial f}{\partial x} = 4x^3 - 3yz$$

$$\frac{\partial f}{\partial y} = -3xz$$

$$\frac{\partial f}{\partial z} = -3xy$$

$$ms\varepsilon = \frac{1}{n}\sum_{i=1}^{n}\underbrace{\left(y_i - (mx_i + b)\right)^2}$$

$$\boxed{\partial/_{\partial m} = \frac{2}{n}\sum_{i=1}^{n} -x_i\left(y_i - (mx_i + b)\right)}$$

$$\partial/_{\partial b} = \frac{2}{n}\sum_{i=1}^{n} -\left(y_i - (mx_i + b)\right)$$

Gradient Descent

The simplest algorithm in the world (almost). Goal:

$$\underset{x}{\text{minimize}}\ f(x)$$

Just iterate

$$\boxed{x_{t+1} = x_t - \eta_t \nabla f(x_t)}$$

where η_t is stepsize.

```python
import numpy as np

def gradient_descent(x,y):
    m_curr = b_curr = 0
    iterations = 1000
    n = len(x)
    learning_rate = 0.001

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        md = -(2/n)*sum(x*(y-y_predicted))
        bd = -(2/n)*sum(y-y_predicted)
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print ("m {}, b {}, iteration {}".format(m_curr,b_curr,i))

x = np.array([1,2,3,4,5])
y = np.array([5,7,9,11,13])
```

x = [1, 2, 3, 4, 5]
y = [3, 7, 9, 11, 13]

-

```python
import numpy as np

def gradient_descent(x,y):
    m_curr = b_curr = 0
    iterations = 10
    n = len(x)
    learning_rate = 0.01

    for i in range(iterations):
        y_predicted = m_curr * x + b_curr
        cost = (1/n) * sum([val**2 for val in (y-y_predicted)])
        md = -(2/n)*sum(x*(y-y_predicted))
        bd = -(2/n)*sum(y-y_predicted)
        m_curr = m_curr - learning_rate * md
        b_curr = b_curr - learning_rate * bd
        print ("m {}, b {}, cost {} iteration {}".format(m_curr,b_curr,cost, i))

x = np.array([1,2,3,4,5])
y = np.array([5,7,9,11,13])
```

```
Run gradient_descent
m 0.62, b 0.18, cost 89.0 iteration 0
m 1.0928, b 0.3192, cost 52.2503999999999 iteration 1
m 1.453232, b 0.42724799999999996, cost 30.831949440000002 iteration 2
m 1.7278860800000002, b 0.5115091199999999, cost 18.347751350784 iteration 3
m 1.9370605952000002, b 0.5776057727999999, cost 11.070010749324897 iteration 4
m 2.096250917888, b 0.6298300216319999, cost 6.826353152519786 iteration 5
m 2.2172859146547204, b 0.6714583661260799, cost 4.350826141683065 iteration 6
m 2.309195511463117, b 0.7049920439242751, cost 2.9056952040975976 iteration 7
m 2.3788729763057748, b 0.7323404723580026, cost 2.0610450731046615 iteration 8
m 2.431580493177024, b 0.7549612843324961, cost 1.5663423003130599 iteration 9

Process finished with exit code 0
```

-

♦ Basic code

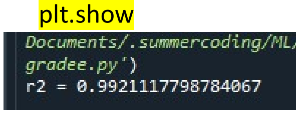
```python
import numpy as np
import matplotlib.pyplot as plt
x=np.array([1,2,3,4,5])
y=np.array([5,7,9,11,13])
m_curr=float(0)
b_curr=float(0)
lr=0.02#learning_rate
n=len(x)
itr=200#no of iterations
plt.scatter(x,y)
cost=[]  #difference of acutal and predicted
for i in range(itr):#interating  itr times
    y_pred=m_curr*x+b_curr
    #prediction in every interation
```

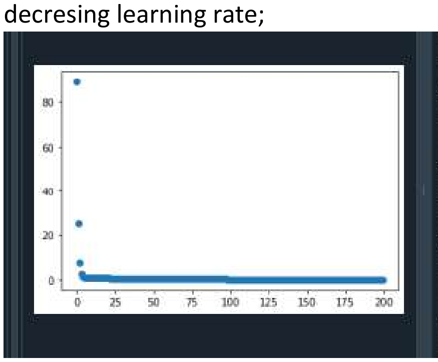
```
    cost_tmp=(1/n)*sum([val**2 for val in (y-y_pred)])
    #calculationg difference
    cost.append(cost_tmp)

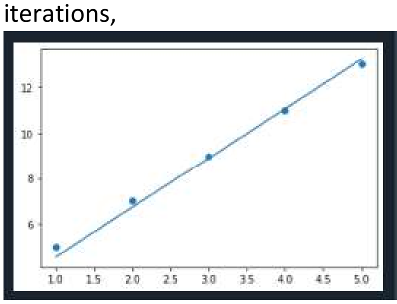
    dm=-(2/n)*sum(x*(y-y_pred))
    db=-(2/n)*sum(y-y_pred)
    m_curr=m_curr-lr*dm
    b_curr=b_curr-lr*db

    #print("m {},b {}, cost {}, iteration {}".format(m_curr,b_curr,cost_tmp,i))
plt.plot(x,y_pred)

from sklearn.metrics import r2_score
r2=r2_score(y, y_pred)
print("r2 =",r2)
plt.show()
plt.figure
index=np.arange(200)
plt.scatter(index,cost)
plt.show
```

- 

can increase it by increasing iterations and decresing learning rate;
- 

#cost array , decreases as increase in iterations,
- 

#final prediction line

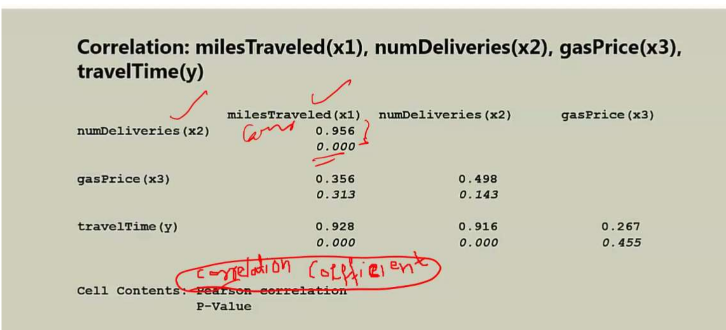
Article

A Prediction Algorithm for Paddy Leaf Chlorophyll Using Colour Model Incorporate Multiple Linear Regression

Sattarpoom Thaipanit[a] and Mahasak Ketcham[b,*]

Department of Information Technology Management, King Mongkut's University of Technology North Bangkok, 1518 Pibulsongkram Road, Bangsue, Bangkok 10800, Thailand
E-mail: [a]sattarpoom@hotmail.com, [b]mahasak.k@it.kmutnb.ac.th (Corresponding author)

- Multiple linear regression
  - Home prices => area bedrooms age price
    - Price=m1*area+m2*bedrooms+m3*age+b
  - Fuel economy=vehicle,engine displavement,horsepower,type of transmission
  - Overfitting decreases the correlation
  - Multi collinearity increases correlation
  -



Correlation: milesTraveled(x1), numDeliveries(x2), gasPrice(x3), travelTime(y)

	milesTraveled(x1)	numDeliveries(x2)	gasPrice(x3)
numDeliveries(x2)	0.956 *0.000*		
gasPrice(x3)	0.356 *0.313*	0.498 *0.143*	
travelTime(y)	0.928 *0.000*	0.916 *0.000*	0.267 *0.455*

Cell Contents: ~~Pearson correlation~~
P-Value

  -
  - Company profit using label and one hot encoders to separate the sates

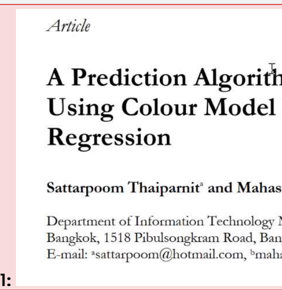
```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 15 11:18:42 2021

@author: Mr.BeHappy
"""

import pandas as pd;
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder

companies=pd.read_csv('C:/Users/rajar/Documents/.summercoding/ML/to
train/profit prediction/1000_companies.csv')
```

Commented [RR1]:



Article

A Prediction Algorithı Using Colour Model I Regression

Sattarpoom Thaipanit[a] and Mahasal

Department of Information Technology M Bangkok, 1518 Pibulsongkram Road, Bang E-mail: [a]sattarpoom@hotmail.com, [b]mahas

Commented [RR2]: Examples

```
data=companies
companies.head()
x=companies['R&D Spend'].values.reshape(-1,1)
y=companies['State'].values.reshape(-1,1)
from sklearn.preprocessing import LabelEncoder
from sklearn.compose import ColumnTransformer
le=LabelEncoder()
data.State=le.fit_transform(data.State)
columnTransformer = ColumnTransformer([('encoder',OneHotEncoder(),[3])],remainder='passthrough')

data = np.array(columnTransformer.fit_transform(data), dtype = np.float64)

#extracting features
X=data[:,:-1]
#extracting targets
Y=data[:,-1]
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.3,random_state=0)

lin_reg=LinearRegression()
lin_reg.fit(X_train,y_train)
y_pred=lin_reg.predict(X_test)
print("coeff:",lin_reg.coef_)
print("intercept:",lin_reg.intercept_)
from sklearn.metrics import r2_score
score=r2_score(y_pred,y_test)
print('prediction accuracy:',score)
import statsmodels.api as sm
X = sm.add_constant(X)
model= sm.OLS(Y, X).fit()
model.summary()
```

-
```
coeff: [ 4.46921768e+02 -3.42694235e+02 -1.04227533e+02  5.26047095e-01
  9.78530820e-01  9.80946128e-02]
intercept: -66123.76082364793
prediction accuracy: 0.9239867538223704
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.950
Model:                            OLS   Adj. R-squared:                  0.950
Method:                 Least Squares   F-statistic:                     3769.
Date:                Sat, 19 Jun 2021   Prob (F-statistic):               0.00
Time:                        15:42:36   Log-Likelihood:                -10588.
No. Observations:                1000   AIC:                         2.119e+04
Df Residuals:                     994   BIC:                         2.122e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const      -5.263e+04   2977.655    -17.674      0.000   -5.85e+04   -4.68e+04
x1         -1.743e+04   1087.818    -16.019      0.000   -1.96e+04   -1.53e+04
x2         -1.787e+04   1083.657    -16.492      0.000      -2e+04   -1.57e+04
x3         -1.733e+04   1074.779    -16.122      0.000   -1.94e+04   -1.52e+04
x4             0.5531      0.035     15.892      0.000       0.485       0.621
x5             1.0262      0.031     33.014      0.000       0.965       1.087
x6             0.0811      0.017      4.814      0.000       0.048       0.114
==============================================================================
Omnibus:                     1577.782   Durbin-Watson:                   1.690
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1032877.856
Skew:                           9.327   Prob(JB):                         0.00
Kurtosis:                     159.336   Cond. No.                     2.53e+18
==============================================================================
```

- IRIS dataset, predictiong the species;

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

leaves=pd.read_csv('C:/Users/rajar/Documents/.summercoding/ML/to
train\iris/Iris.csv')
data=leaves
leaves.head()
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.compose import  ColumnTransformer

le=LabelEncoder()
data['Species']=le.fit_transform(data['Species']);
columnTransformer=ColumnTransformer([('encoder',OneHotEncoder(),[4])],re
mainder='passthrough')

data=np.array(columnTransformer.fit_transform(data), dtype = np.float64)

X=data[:,:-1]
Y=data[:,-1]
#extracting targets
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test                                         =
train_test_split(X,Y,test_size=0.3,random_state=0)

lin_reg=LinearRegression()
lin_reg.fit(X_train,y_train)
y_pred=lin_reg.predict(X_test)
print("coeff:",lin_reg.coef_)
print("intercept:",lin_reg.intercept_)
from sklearn.metrics import r2_score
score=r2_score(y_pred,y_test)
print('prediction accuracy:',score)
import statsmodels.api as sm
X = sm.add_constant(X)
model= sm.OLS(Y, X).fit()
model.summary()
```

▪

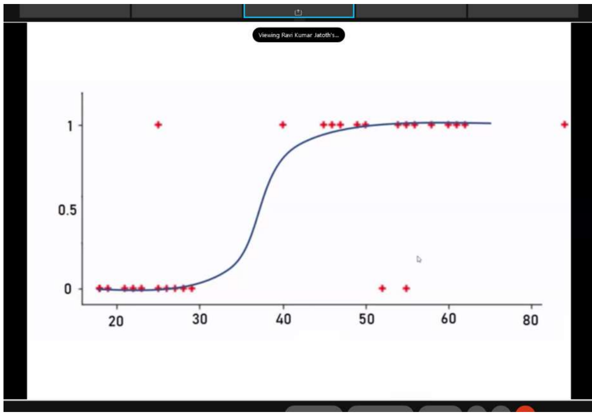
```
In [67]: runcell(0, 'C:/Users/rajar/Documents/.summercoding/ML/to train/iris/untitle
coeff: [-0.48089423  0.06162781  0.41926642 -0.1030543   0.21940979  0.27963455]
intercept: 0.07655739532205819
prediction accuracy: 0.9326560643682853
```

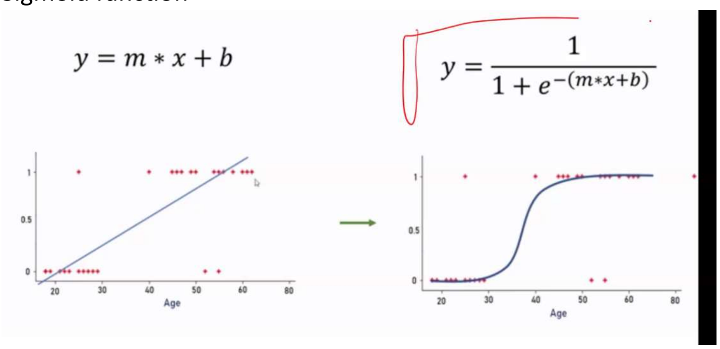
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.954
Model:                            OLS   Adj. R-squared:                  0.953
Method:                 Least Squares   F-statistic:                     599.5
Date:                Sat, 19 Jun 2021   Prob (F-statistic):           1.79e-94
Time:                        15:45:58   Log-Likelihood:                 59.398
No. Observations:                 150   AIC:                            -106.8
Df Residuals:                     144   BIC:                            -88.73
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0632      0.123      0.515      0.607      -0.179       0.305
x1            -0.5509      0.097     -5.666      0.000      -0.743      -0.359
x2             0.1073      0.051      2.106      0.037       0.007       0.208
x3             0.5068      0.090      5.639      0.000       0.329       0.684
x4            -0.0948      0.045     -2.129      0.035      -0.183      -0.007
x5             0.2497      0.048      5.248      0.000       0.156       0.344
x6             0.2409      0.049      4.947      0.000       0.145       0.337
==============================================================================
Omnibus:                        5.957   Durbin-Watson:                   1.840
Prob(Omnibus):                  0.051   Jarque-Bera (JB):                8.404
Skew:                          -0.159   Prob(JB):                       0.0150
Kurtosis:                       4.115   Cond. No.                     2.77e+16
==============================================================================
```

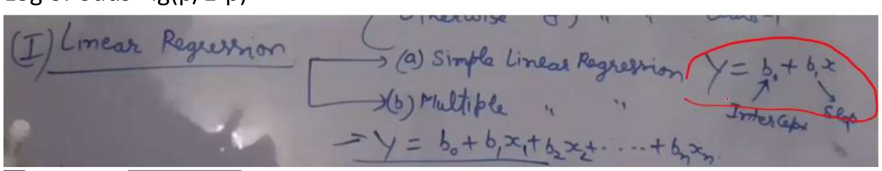
▪

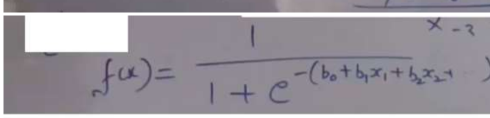
▪

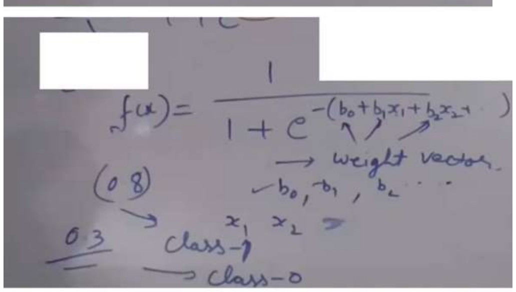
- 18-06
  - Intro to logical regression
  - LOGISTIC REGRESSION
  - Rat obesity :

- 

#sigmoid curve;
- Sigmoid function


- Odds=ratio of favour and not in favour;p/1-p
  - For generalisation we take log,,
  - Log of odds= lg(p/1-p)
  - 

  - 

  - 

  - If f is >0.5 then odds are in favour else not in favour;

- $$f(x) = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + b_2 x_2)}}$$

  $f(x) =$ probability $(x_1, x_2) \rightarrow$ Class-0, Class-1

  $$P(y=1 \mid x ; w) = f(x)$$

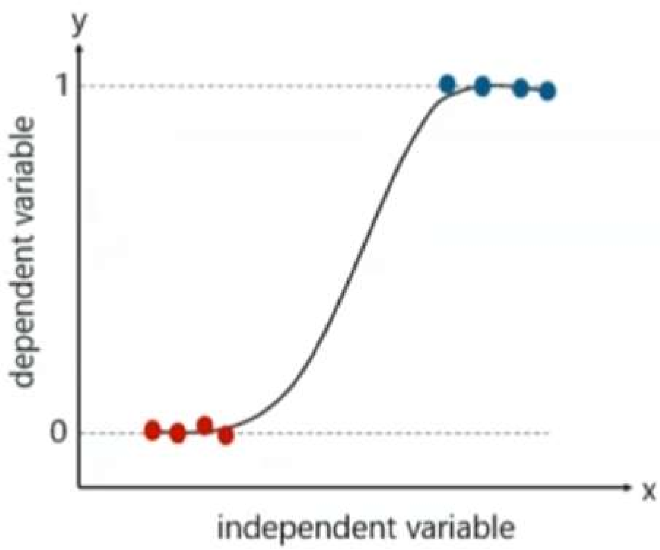
  $$P(y=0 \mid x ; w) = 1 - f(x)$$

  $$P(y \mid x ; w) = (f(x))^{y} (1 - f(x))^{(1-y)}$$

- m - Independent training samples

  $$\sum_{i=1}^{m} P(y^i \mid x^i ; \underline{w}) = \sum_{i=1}^{m} \left[(f(x^i))^{y^i} (1 - f(x^i))^{(1-y^i)} \right] = L(b)$$

- We get a sigmoid curve;



- Basic code on advertising

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 18 11:02:24 2021

@author: Mr.BeHappy
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from math import exp
data=pd.read_csv('C:/Users/rajar/Documents/.summercoding/ML/to
train/socialnetworkadd/Social_Network_Ads.csv')
data.head()
plt.scatter(data['Age'], data['Purchased'])
plt.show()
X_train, X_test, y_train, y_test = train_test_split(data['Age'],
data['Purchased'], test_size=0.20)

def normalize(X):
   return X - X.mean()
def predict(X, b0, b1):
   return np.array([1 / (1 + exp(-1*b0 + -1*b1*x)) for x in X])

def logistic_regression(X, Y):
   X = normalize(X)
   b0 = 0
   b1 = 0
   L = 0.01
   epochs = 500
   for epoch in range(epochs):
      y_pred = predict(X, b0, b1)
      D_b0 = -2* sum((Y - y_pred) * y_pred * (1 - y_pred))  # Derivative
of loss wrt b0
      D_b1 = -2* sum(X * (Y - y_pred) * y_pred * (1 - y_pred))  #
Derivative of loss wrt b1
      b0 = b0 - L * D_b0
      b1 = b1 - L * D_b1
   return b0, b1

# Training the model
b0, b1 = logistic_regression(X_train, y_train)
X_test_norm = normalize(X_test)
y_pred = predict(X_test_norm, b0, b1)

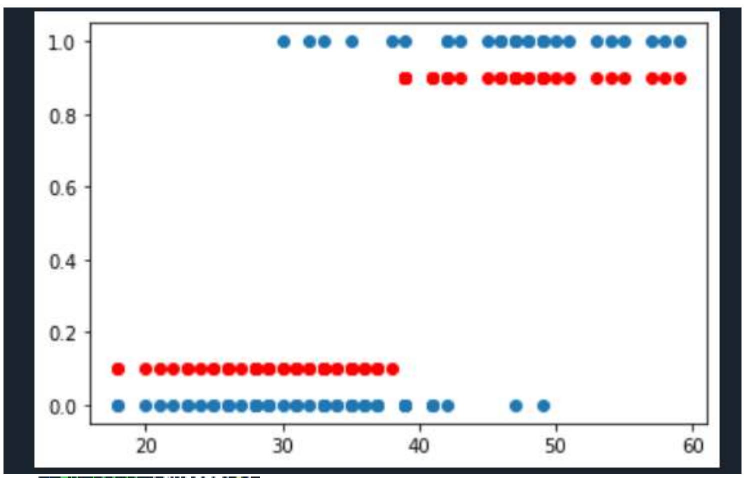
y_pred = [0.9 if p >= 0.5 else 0.1 for p in y_pred]
plt.clf()
plt.scatter(X_test, y_test)
plt.scatter(X_test, y_pred, c="red")
# plt.plot(X_test, y_pred, c="red", linestyle='-', marker='o') # Only if
values are sorted
```

```
plt.show()

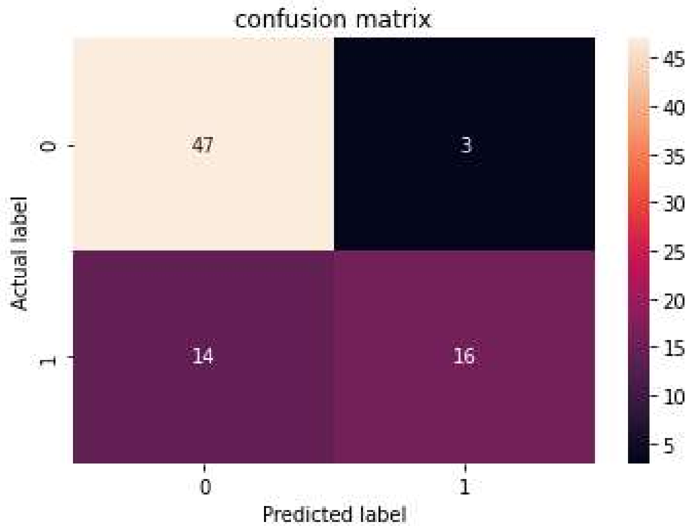
from sklearn import metrics

import seaborn as sns

y_pred = [1 if p >= 0.5 else 0 for p in y_pred]
conf_matrix=metrics.confusion_matrix(y_test,y_pred)
print(conf_matrix)
sns.heatmap(conf_matrix,annot=True)
plt.title('confusion matrix')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
```




```
[[47  3]
 [14 16]]
Accuracy: 0.7875
```



- Obese vs non obese
  - Code

```
import numpy as np

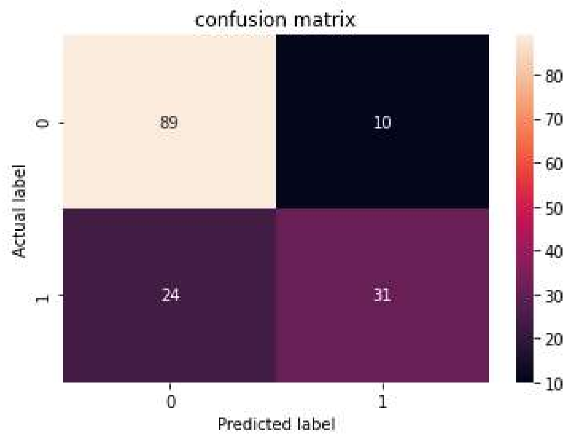
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn import metrics

import seaborn as sns
data=
pd.read_csv('C:/Users/rajar/Documents/.summercoding/ML/to
train/diabetis/diabetes.csv')
dd=pd.DataFrame({ 'hi':data['Pregnancies']},index=data['Glucose'])
dd.describe()
X=data.iloc[:,:-1]
y=data.iloc[:,-1]

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,ran
dom_state=1)
logreg=LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(df)
```

```
conf_matrix=metrics.confusion_matrix(y_test,y_pred)
print(conf_matrix)
sns.heatmap(conf_matrix,annot=True)
plt.title('confusion matrix')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
```

- ```
  684       0          0
  643       0          0

  [154 rows x 2 columns]
  [[89 10]
   [24 31]]
  Accuracy: 0.7792207792207793
  ```
- 
  ```