UNIT-V Regression using MLP

- independent variables - i/p variables
- dependent variables - o/p variables.

Regression -

- It is a technique used to determine relationship b/w independent and dependent variables.
- 3 techniques -

① Simple linear regression:

- determines relationship between independent & dependent variables

$$y = B_0 + B_1 x$$

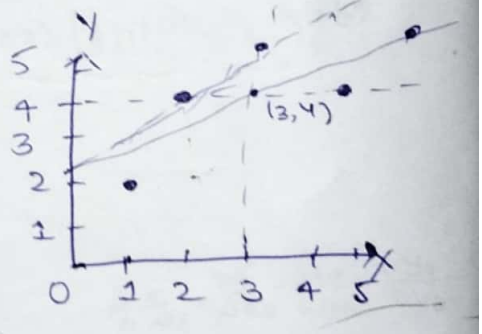
x - independent variable
y - dependent variable
B_0 - y intercept
B_1 - slope

Eg:

x	y	$x-\bar{x}$	$y-\bar{y}$	$(x-\bar{x})(y-\bar{y})$	$(x-\bar{x})^2$	$(y-\bar{y})^2$
1	2	-2	-2	4	4	4
2	4	-1	0	0	1	0
3	5	0	1	0	0	1
4	4	1	0	0	1	0
5	5	2	1	2	4	1

$\bar{x} = 3$ $\bar{y} = 4$

$$B_1 = \frac{\sum (x-\bar{x})(y-\bar{y})}{\sum (x-\bar{x})^2} = \frac{6}{10} = 0.6$$



$$\bar{y} = B_0 + B_1 \times \bar{x}$$

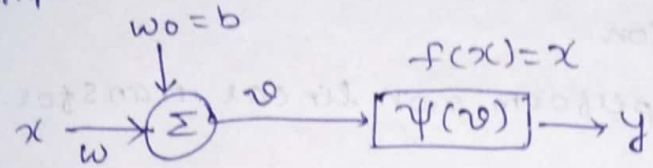
$$4 = B_0 + 0.6 \times 3$$

$$B_0 = 4 - 0.6 \times 3$$

$$= 4 - 1.8$$

$$B_0 = 2.2$$

Implementation using perceptron -



- can be implemented using single layer perceptron with linear activation fn

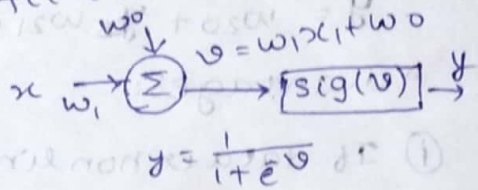
$$v = wx + w_0$$
$$y(v) = wx + w_0$$

Regression & Perceptron -

Logistic Regression:

- uses non-linear line to best fit the data.

- mathematical representation:



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

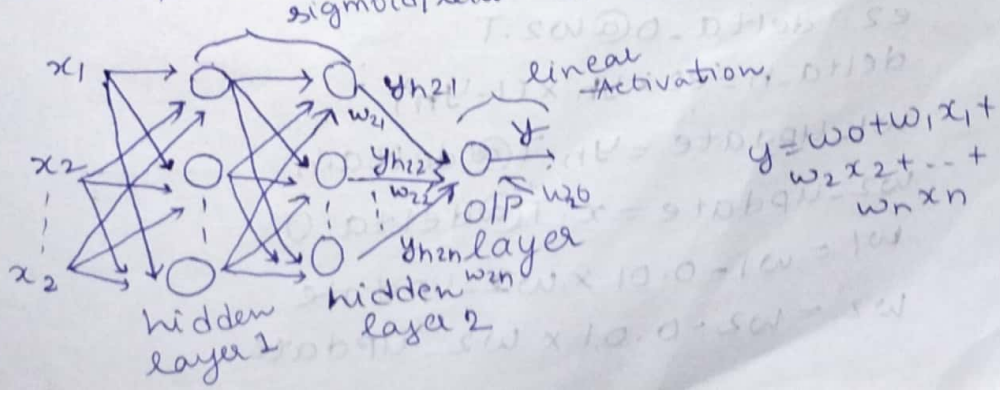
- single layer perceptron can be used to implement logistic regression.
- sigmoid activation function is used as activation function.

multivariant Regression:

- It is used to determine relationship b/w multiple independent, one dependent variable

$$f(x_1, x_2, --, x_n) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + -- + \beta_n x_n$$

- Although single layer perception can accept morethan 1 input, it is applicable only to linear separable.



- multilayer perception is best suitable to implement multivariat regression.
- hidden layers may perform non linear transformation.
- output layer neurons (can perform linear transformation).
- activation functions:
  - at hidden layer
    - Relu, sigmoid
  - at output layer
    - linear activation function.

$y = w_{20} + w_{21} \cdot y_{h21} + w_{22} \cdot y_{h22} + \cdots + w_{2n} \cdot y_{h2n}$

Advantages:

① If data is non linear separable, MLP transforms data into linear seperable then applies linear transformation.

② MLP improves prediction accuracy.

③ make predictions even if data has noise.

21/10/22

Implementing Regression

```
def train(x,y):
    #forward phase
    vh = x@w1
    yh = sig(vh)
    vo = yh@w2
    yo = ide(vo)
    #back Propagation Phase
    eo = yo - y
    delt_o = e
    e2 = delta_o@w2.T
    delta_h = e2 x yh x (1-yh)
    w1_update = yh.T@delta_h
    w2_update = x.T@delta_o
    w1 = w1 - 0.01 x w1_update
    w2 = w2 - 0.01 x w2_update.
```

22/10/22

Implementing Regression using sklearn:

→ neural_network: - It provides 2 classifiers

→ MLPClassifier - classification
→ MLPRegressor - regression

MLP Regressor - create multi layer perceptron by calling MLPRegressor()

→ * hidden_layer_sizes:

- tuple
- default → (100,)
- custom sizes - pass a tuple with sizes of reqn no. of hidden layers.

eg: for 3 hidden layers

(10, 7, 4)

- activation -
  - identity
  - relu
  - sigmoid
  - tanh

- solver - specifies optimization (specifies Optimization Algorithm)
  - specifies ~~lbfgs~~
  - adam

- tol (stopping criteria)
  - tells when to stop running algorithm

use default value = $1e^{-3}$

eg:

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
d = datasets.fetch_california_housing()
# set i/p & o/p variables
x = d.data
y = d.target
```

```
#create training & test datasets
xtr, xte, ytr, yte = train_test_split(x, y,
  random_state = True, test_size = 0.2)
#create model
mlpr = MLPRegressor(hidden_layer_sizes = (8, 6))
mlpr.fit(xtr, ytr)
mlpr.predict(xte)
mlpr.score(xte, yte)
```

```
from sklearn.linear_model import LinearRegression,
                                 LogisticRegression
lr = LinearRegression()
lg = LogisticRegression()
lr.fit(xtr, ytr)
lr.predict(xte)
lr.score(xte, yte)
```

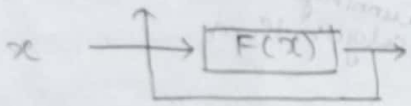
26/10/22

Function Approximation using MLP

⇒ f(x) → d

→ function Approximation

checks the o/p with actual/target o/p



⇒ Given unknown function f(x) which produces o/p y

⇒ Function approximation uses known function f(x) and try to produce o/p

$y \approx d$

⇒ MLP with single hidden layer enough to approximate any function.

Universal approximation theorem

⇒ It is enough to have MLP with single hidden layer to with approximate any function.

m_0 - size of input

m_1 - Size of hidden layer

w_{1x} - weight vectors of hidden layer, o/p layer

$f()$ - unknown function

$f(x)$ = known function

$F(x) = \alpha v(w^T x + b)$ $|f(x) - F(x)| < \epsilon_1$

ϵ_1 Error accepted Value

Bounds on the error:

$\hat{f}(w)$ fourier transform

inverse formula

$$f(w) = \int_{R^{m_0}} \hat{f}(w) \exp(jw^T x)\, dw$$

$$cf = \int_{R^{m_0}} \hat{f}(w) \times \|w\|^2 dw$$

$$R = \frac{1}{2} \sum_{i=1}^{n} (f(x_i) - f(x_i))^2 < c.f$$

$$R = \frac{1}{N} \sum_{i=1}^{n} (f(x_i) - f(x_i))^2 < \frac{cf}{m_1}$$

Low & cost depends of m_1

Curse on dimensionality:

Covers Theorem

→ Non linear Separable, data can be classified by transforming data in to linear seperable form.

Ex:- binary classification

(or)

dichotomy

transform → $\phi(x)$ ⟶ Radial Basics functions

$w^T \phi(x) = 0$

$w^T \phi(x) > 0, x \in C_1$

$w^T \phi(x) < 0 \quad x \in C_2$

$\sum_{i=1}^{m_1} a_{i_1, i_2, i_3, i_4, \cdots i_m} x_{i_1}^{a_{i_1}} x_{i_2}^{a_{i_2}} \cdots x_{ik}$

→ $x_{i_1}, x_{i_2}, x_{i_3}, \cdots x_{ir}$

Called as monomial → $\frac{(m_0 - r)!}{m_0! r!}$.

Types of Radial Basis functions Networks

There are 3 types

① Gaussian function

$$\phi(x) = \exp\left(\frac{-r^2}{2\sigma^2}\right)$$

$r \rightarrow$ radius

$\sigma \rightarrow$ spread

② Multi Quadratic Radial Basis function

$$\phi(x) = \sqrt{r^2 + c^2} \quad r \in R$$

c is constant

③ Inverse Multi Quadratic RBF

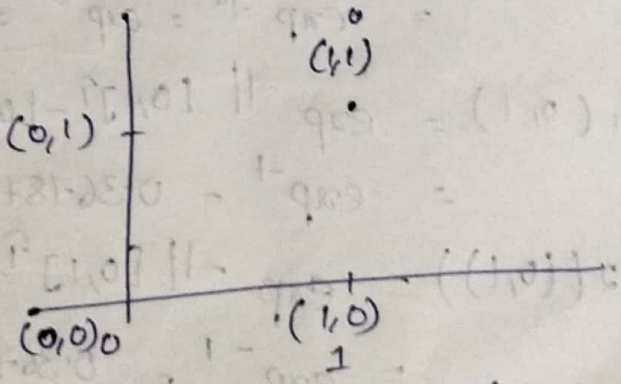
$$\phi(x) = \frac{1}{\sqrt{r^2 + c^2}}$$

Working of RBF Networks

- Similar to the KNN Algorithm
- given a point a distance from center to the point is computed (i.e r)
- Radial basis functions use r to perform transformations

XOR Problem

x_1	x_2	
0	0	0
0	1	1
1	0	1
1	1	0



$\phi(x) = \exp\left(\frac{-r^2}{2\sigma^2}\right)$ - Euclidian distance from the Center point

$2\sigma^2 = 1$

$= \exp^{-x^2}$

$= \exp^{-\|x-t\|^2}$ t is Center

- 2 hidden layer neurons to perform transformation

$\Rightarrow$ 2 centers

$t_1 = [0,0]$

$t_2 = [1,1]$

Curse on dimensionality ... — Prior knowledge

As the I/p data dimensionality is

x_1	x_2	$\phi_1(x)$	$\phi_2(x)$
0	0	1	0.13533
0	1	0.36787	0.36787
1	0	0.36787	0.36787
1	1	0.13533	0

x t_1
x_1 0 0
x_2 0 0

$\sqrt{(0-0)^2+(0-0)^2} = 0$

$\phi_1(x) = \exp^{-\|x-t_1\|^2}$

$\phi_2(x) = \exp^{-\|x-t_2\|^2}$

$\phi_1((0,0)) = \exp^{-\|[0,0]^T-[0,0]\|^2}$

$= \exp^{-(0)} = 1$

$\phi_2((0,0)) = \exp^{-\|[0,0]-[1,1]\|^2}$

$= \exp^{-(\sqrt{2})^2} = \exp^{-2} = 0.13533$

$\phi_1(0,1) = \exp^{-\|[0,1]^T-[0,0]^T\|^2}$

$= \exp^{-1} = 0.36787$

$\phi_2((0,1)) = \exp^{-\|[0,1]^T-[1,1]^T\|^2}$

$= \exp^{-1} = 0.36787$

$\phi_1(1,0) = \exp^{-\|[1,0]^T-[0,0]\|^2}$

$= 0.36787$

$\phi_2(1,0) = \exp^{-\|[1,0]^T-[1,1]\|^2}$

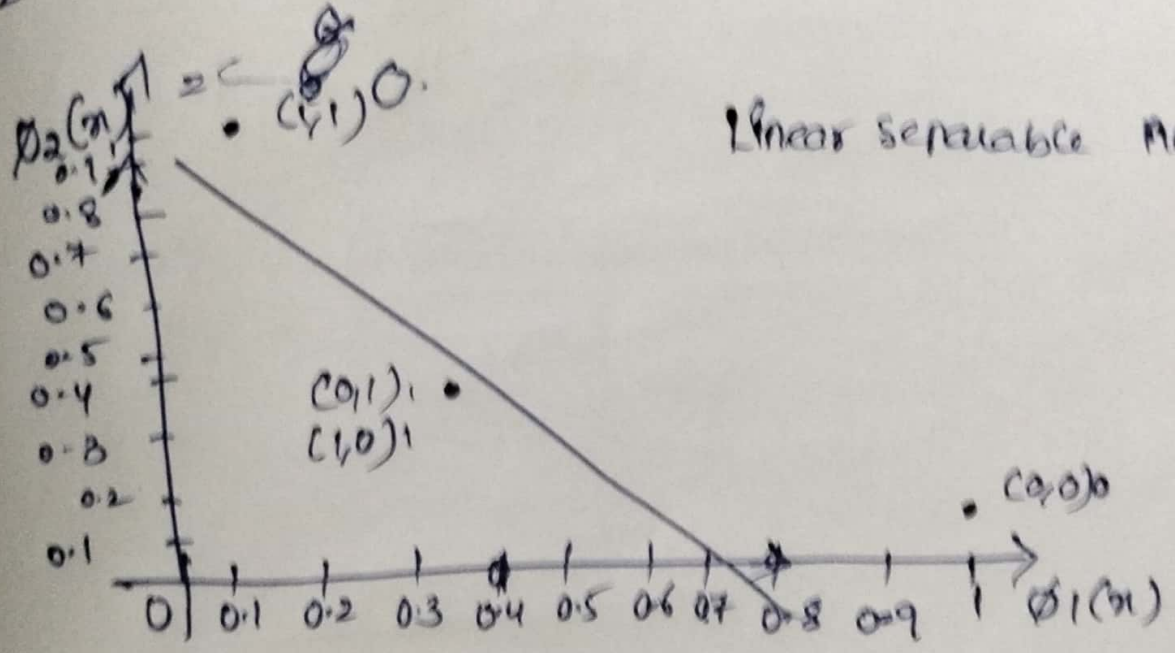
$= 0.36787$

$\phi_1(1,1) = \exp^{-\|[1,1]-[0,0]\|^2}$

$= 0.13533$

$\phi_2(1,1) = \exp^{-\|[1,1]-[1,1]\|^2}$

$= 0$



Linear separable now.

⇒ hidden layers are responsible for transformations

No. of hidden layers = No. of RBS fun's