13/10/22
Thursday

Date
Page

Linear Regression — It is a machine learning algorithm based on Supervised Learning.

It is a Statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variable such as Cost, age, Sales, Temperature, Product price, etc.

Types —

i) Linear Regression with Single values.
ii) Linear Regression with Multiple values.

(-x-) for Single variable

demo ① —
```
>>> import pandas as pd
>>> import seaborn as sns
>>> from sklearn import linear_model. (# for line
```

demo ② —
```
>>> data1 = pd.read_csv(r'Path')
```

	age	premium
0	25	18,000
1	30	32000
2	35	40,000
3	40	47,000
4	45	55,000

} whole data set

Linear Regression :-

Only the main code -

(1) >>> import statsmodels.formula.api as smf
>>> lm = smf.OLS(formula = "dependent variable ~ feature1 + feature2 + ...", data = dataset1).fit()
>>> lm.summary()

(2) # Using Standard Scaler -

>>> x = df.iloc[:, :-1]
>>> y = df.iloc[:, -1]

>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()

>>> scaler.fit_transform(x, y)
or
>>> avr = scaler.fit_transform(x, y)

Using Lasso, Ridge & elastic net in linear Reg.

① >>> from sklearn.linear_model import Ridge, Lasso, ElasticNet
>>> from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV

② >>> lassocv = LassoCV(cv=5, max_iter=20000, normalize=True)
>>> lassocv.fit(x_train, y_train)

③ >>> lassocv.alpha_
>>> lasso = Lasso(alpha = lassocv.alpha_)
>>> lasso.fit(x_train, y_train)

④ >>> lasso.score(x_test, y_test)

∴ as we use the standardisation so before predict we have to transform it.

>>> t = scaler.transform([[1, 2, 3, ..., 577])
>>> lasso.predict(t)

— Ridge —

⑤ >>> ridgecv = RidgeCV(alphas=(0.1, 1.0, 10.0), cv=10, normalize=True)
>>> ridgecv.fit(x_train, y_train)

alpha & lambda are same things

⑥ >>> ridgecv.alpha_
>>> ridge_lr = Ridge(alpha = ridgecv.alpha_)
>>> ridge_lr.fit(x_train, y_train)

⑦ >>> ridge_lr.score(x_test, y_test)
(predict will be same)

Elastic Net —

⑧ >>> elastic = ElasticNetCV(alphas = None, cv = 10)
>>> elastic.fit(x_train, y_train)

>>> elastic.alpha_ (# it is lambda)

>>> elastic.l1_ratio (# it is actual alpha of elastic net)
>>> elastic.l1_ratio_

⑨ >>> elastic_lr = ElasticNet(alpha = elastic.alpha_, l1_ratio = elastic.l1_ratio_)
>>> elastic_lr.fit(x_train, y_train)

⑩ >>> elastic_lr.score(x_test, y_test)

Note - To predict the value of the train dataset / or the user input remember that that if we use Standardization (Standard-Scaler) we have to Transpose the input.

```
>>> t = scaler.transform([[337, 118, 4, 4.5 ..., 111]])
>>> elastic_lr.predict(t)
```

— To export the model in pkl —

```
>>> import pickle
>>> pickle.dump(elastic_lr, open("elastic_model.pkl", 'wb'))
```

```
[ wb =
  rb =
```

LR $y = m_1 x_1 + m_2 x_2 \cdots + c$

m = coef / slope = θ_1,

c = intercept = θ_0

$y = mx + c$ or

$y = \beta_1 x + \beta_0$

16/10/22
Sunday

Internship

Date
Page

Logistic Regression (Binary classification)

11 **Logistic Regression:** — It is a machine learning algorithm based on Supervised learning.

It is a Statistical method that is used for predicting probability of target variable.

Logistic Regression makes probability for classification problems that are discrete in nature.

Ex — English or Hindi, True or false, 1 or 0, Right or wrong, cat or dog or goal.

Types of Logistic Regression —

i) **Binary classification** — Ex:- win or loss, dead or alive.

ii) **Multiclass classification** — Ex:- Onion or Potato or Sweet Potato. Lily or Sunflower or Rose

∴ **Real life example** of logistic Regression are used in ~~hospital~~ hospital to check ~~whether~~ what is the chance of getting heart attack of patients.

↳ Heart Attack Prediction / Tumour Prediction

- Logistic Regression -

In Logistic Regression we Solve Classification Problem.

OVO → One Verse One → Binary Classification
OVR → One Verse Rest → Multi-Class Classification

If your dataset is Skewness that it is generally a good idea to replace Nan value or (any value or '0') in mean

```
>>> df["BM"] = df["BMI"].replace(0, df["BMI"]
                .mean())
```

If there are skewness in the data it mean the colums has a "Outliers"

If there is so much Outlier in a Single feature that we will try to ~~remo~~ reduced the data from that perticular feature.

o drop 10% data.

```
>>> q = df["Insulin"].quantile(.90)
>>> df_new = df[df["Insulin"] < q]
↳ # This is new data set after dropings
```

Q How to remove Outlier?

Ans → To experiment with quantile range we can remove outlier. as it one of the way.

```
>>> q = df["Insulin"].quantile(.95)
>>> df_new = df[df["Insulin"]<q]
```

↓ after this

```
>>> q = df_new["Age"].quantile(.95)
>>> df_new = df_new[df_new["Age"]<q]
```

(Quatile is to drop the data from data set to remove Outliers)

Q Do we do Regularization in Logistic Regression?

Ans Yes we do Regularization in Logistic Regression but by default.

If

- solver → "lbfgs" → that it do L2 regula. by default.

and

- solver = 'saga' → that it is elastic net (L1+L2) regularization.

- Solver = "liblinear" → It support both L1 and L2 reg. with a dual formulation only for the L2 Penalties

Note → For Small dataset, "liblinear" is a good choice, whereas 'sag' & 'saga' are faster for large ones.

→ liblinear is limited to one-versus-rest

→ We will not use "liblinear-Solver" for multiclass classification

Logistic Regression work on Sigmoid function -

$$Sig(n) = \frac{1}{1+e^{-n}}$$

If we feed an output value to the Sigmoid function, it will return the probability of the outcome between 0 & 1.

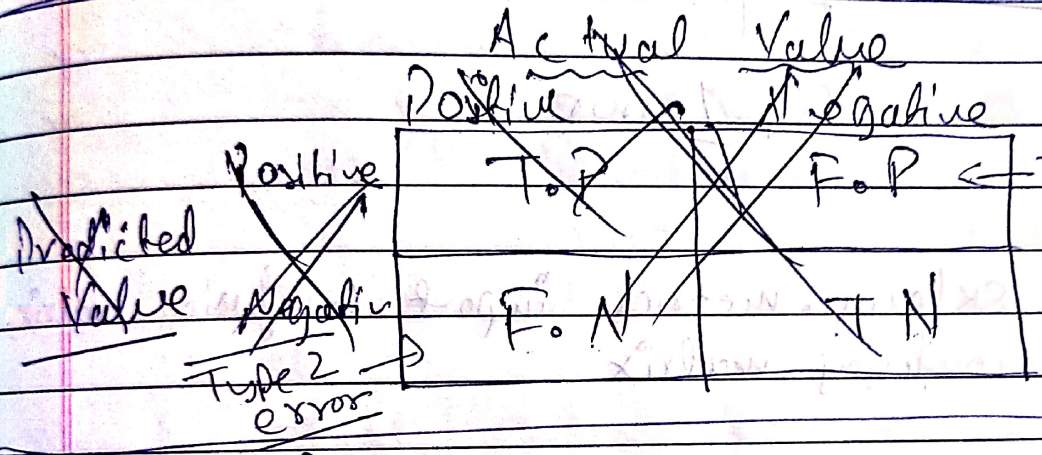
↓↓

If the value is below 0.5, then the output returns as No / Fail. If it above 0.5 that means the output is returned as Yes / Pass.

Parameters

Verbose = Integer that we want to see.

Confusion Matrix :- used in order to evaluate the performance of a classification Model
It is N * N matrix



- when Predicted Value Matches with Actual → T.P
- when Value is false & model is also saying false → T.N
- when the Actual Value is false & Predicted Value Saying True → F.P (Type 1 - error)
- when the Actual Value is True & model predicts False → F.N

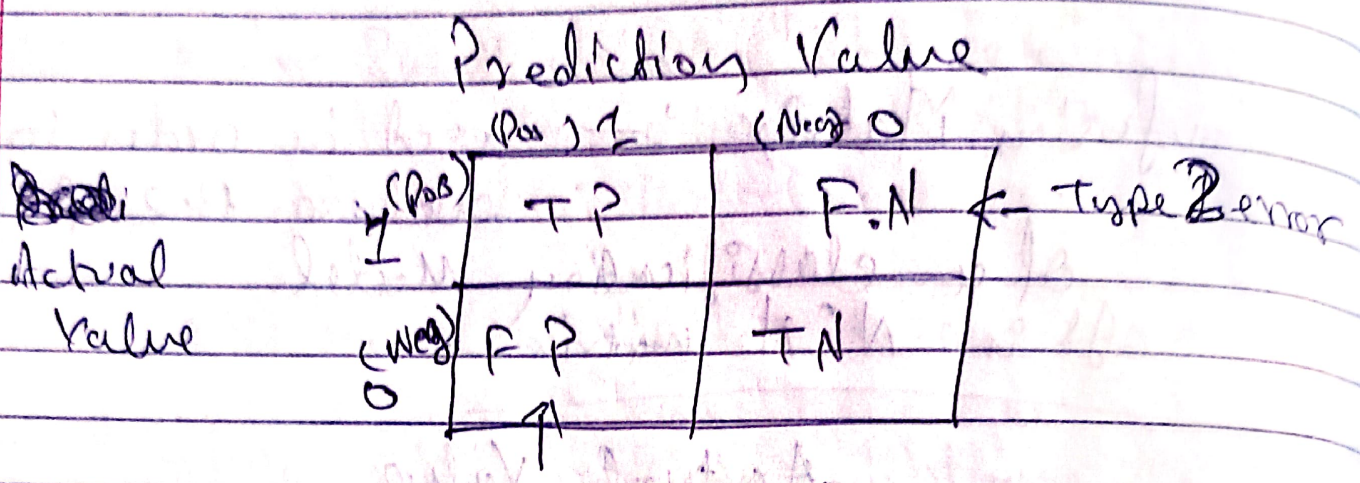
$$\text{Accuracy} = \frac{TP + TN}{\text{Total}}$$

$$\text{Precision} = \frac{TP}{\text{(Pred. Yes)}\ TP + FP}$$

$$\text{recall} = \frac{TP}{\text{(actual yes)}\ FN + TP}$$

Error Rate = 1 - accuracy
or

$$\left(\frac{\text{Type1 error} + \text{Type2 error}}{\text{Total}}\right) \quad \frac{FP + FN}{\text{Total}}$$




```
>>> from sklearn.metrics import confusion_matrix,
    plot_confusion_matrix

>>> from sklearn.linear_model import LogisticRegressio

>>> logr = LogisticRegression(verbose=1)
>>> logr_liblinear= LogisticRegression(Verbose=1,solver="liblinea

>>> logr.fit(x_train, y_train)
>>> logr_liblinear.fit(x_train, y_train)

>>> y_pred = logr.Predict(x_test)
>>> y_pred_liblinear = logr_liblinear.predict(x_test)

>>> confusion_matrix(y_test, y_pred)
>>> confusion_matrix(y_test, y_pred_liblinear)
```

Now we will make function -

```
>>> def model_eval(y_test, y_pred):
>>>     tn, fp, fn, tp = confusion_matrix(y_test,
                                y_pred).ravel()
>>>     accuracy = (tp+tn)/(tp+tn+fp+fn)
>>>     precision = tp/(tp+fp)
>>>     recall = tp/(tp+fn)
>>>     result = {"accuracy": accuracy, "precision":
                    precision, "recall": recall}
>>>     return result
```

```
>>> model_eval(y_test, y_pred)
>>> model_eval(y_test, y_pred_liblinear)
```

— ROC AUC Score —

The Roc AUC Score tells us how efficient the model is. The higher the AUC, the better the model performance

```
>>> from sklearn.metrics import roc_curve, roc_auc_score
```

```
>>> roc_auc_score(y_test, y_pred)
>>> roc_auc_score(y_test, y_pred_liblinear)
```

ROC & AUC (Threshold = [0, 0.2, 0.4, 0.6, ..., 1])

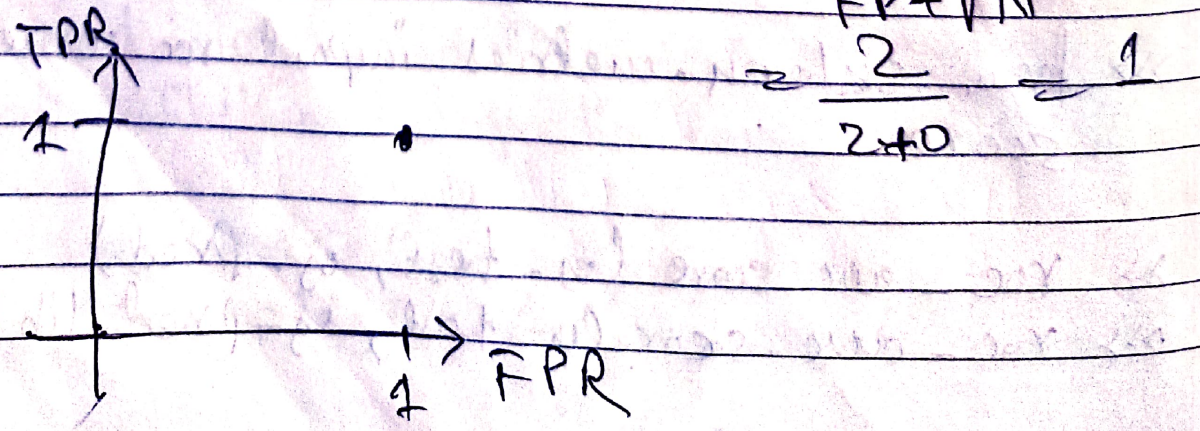
(1) → y (actual value)	$\hat{y}$	$\hat{y}$(0)
1	0.8	1
0	0.96	1
1	0.4	1
1	0.3	1
0	0.2	1
1	0.7	1

Predicted \ Actual	1	0
1	TP	FP
0	FN	TN

If threshold = 0

∴ Sensitivity → TPR (True positive Rate) $= \frac{TP}{TP+FN} = \frac{4}{4+0} = 1$

∴ Specificity → FPR (false positive Rate) $= \frac{FP}{FP+TN} = \frac{2}{2+0} = 1$

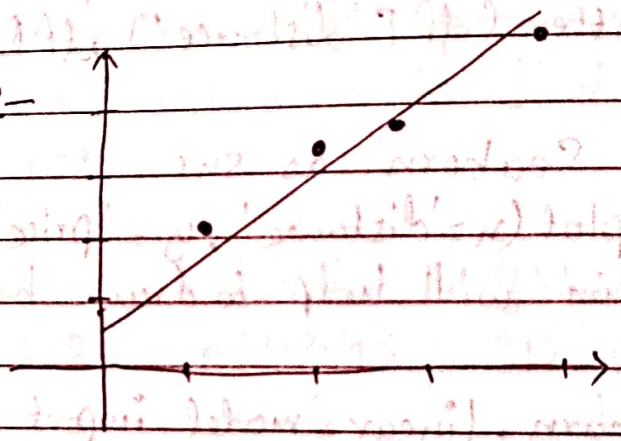


∴ ROC & AUC should be only used for classification problem to desired threshold and how the model performed.

14/10/22
Friday

10 Polynomial Regression :— It is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modeled as an n^{th} degree polynomial.

⇒ linear model :—

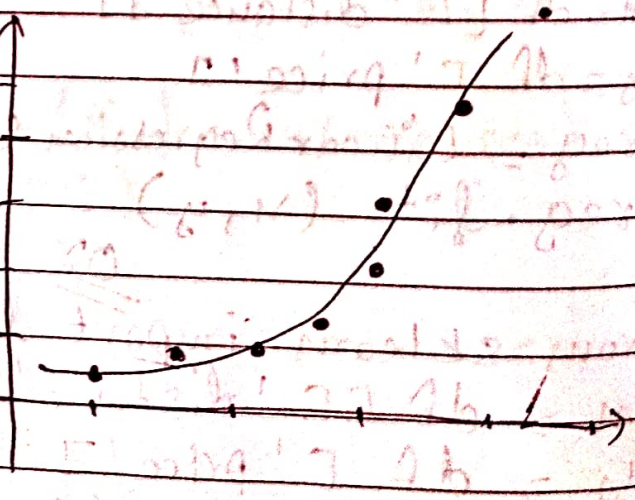


⇒ Polynomial model :—

0 Degree Polyno. :- y = constant

1 Degree Polyno. :- $y = mx^1 + c$

2 degree Polyno. :- $y = ax^2 + bx + c$



Generalized Polynomial Equation

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n$$

Coding ① dataset = df1 = pd.read_csv(r"path")

Dataset

df1 =

	Position	level	Salary
0	Bussiness Analy	1	45,000
1	Junior Cons.	2	50,000
2	Senior Cons.	3	60,000
3	Manager	4	80,000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

Coding ② >>> x = df1.iloc[:, 1:2].values (# independent Variable)
>>> x

output = array([[1],
[2], ...
.....[10]

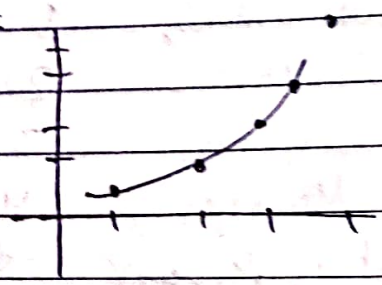
Coding ③ >>> y = df1.iloc[:, 2:].values
>>> y

output = array([[45000],
[50,000], ...
.....[1000000]

Code no ⑤ >>> import matplotlib.pyplot as plt

Code no ⑤ >>> plt.scatter(x, y)

Output:



→ fig-1 (Polynomial)

Code no ⑥ ∴ Now we will check whether the data suit in 'Polynomial regression' or 'linear regression' by drawing scatter plot

>>> sns.lmplot(x='level', y='salary', data=df)

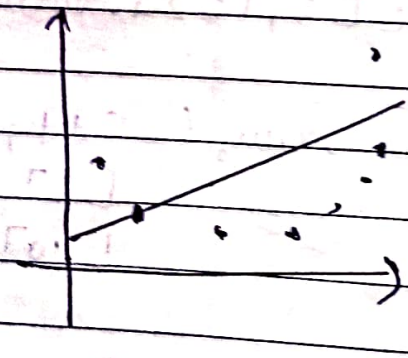


fig-2 (linear regression)

∴ Hence fig-1 is Polynomial Reg. and fig-2 is linear Reg. & it is clear that point data is close best with Polynomial Reg as best fit line touch most value in Polynomial.

import linear regression

```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit(x,y)
```

linear regression

```
>>> reg.predict([[6.5]])
output = 330378.78
```

We already know that linear regression will be false. but we just check it.

∴ hence the linear regression gives the wrong prediction, so we will use polynomial regression.

import Polynomial regression

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(degree=2)
>>> x_poly = poly.fit_transform(x)
>>> reg2 = linear_model.LinearRegression()
>>> reg2
>>> reg2.fit(x_poly, y)
```

(# output will be as LinearRegression() but it is a Poly linear)

>>> reg2.predict(poly.fit_transform([[6.5]]))

output:- array([[189498.1077]])

∴ here the prediction is almost correct.

I.Q 6 In mean why do we have to use Mean Square Error (MSE)

Ans The reason is that we have only one Global Minima no local minima

local minima :-