Lecture-03

→ Practicals [Linear, Ridge & Lasso, Logistic]

→ Naive Baye's Intution.

→ KNN algorithms

Practicals: [Linear Regression & Ridge and Lasso]

Program:

```
from Sklearn.datasets import load-boston

import numpy as np
import pandas as pd
import Seaborn as Sns
import matplotlib.pyplot as plt
% Matplotlib inline

df = load_boston()
type(df)

dataset = pd.Dataframe(df.data)
dataset.Columns = df.features_names
dataset.head()
```

Create dependent feature

```
dataset['price'] = df.target
dataset.head()
```

Dividing the dataset into independent and dependent features

```
x = dataset.iloc[:, :-1]  # independent features
y = dataset.iloc[:, -1]  # dependent features

x.head()
y.head()
```

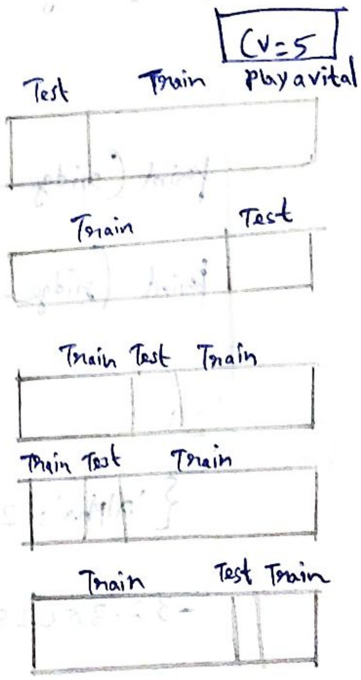
#Linear Regression

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
lin_reg = LinearRegression()
mse = cross_val_score(lin_reg, x, y, scoring = 'neg_mean_squared_error', cv=5)

mse  # return 5 value
```

Output:

[-12.46030, -26.04886, -33.074, -80.762, -33.31360]




```
mean_mse = np.mean(mse)
print(mean_mse)
```

Output: -37.1318074

To predict with test data.

```
lin_reg.predict()   # To pass the test data to make
                      prediction.
```

Ridge Regression

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
```

GridSearchCV play among all (α) parameter to give best (α) value for our model.

```
ridge = Ridge()
params = {'alpha': [1e-15, 1e-10, 1e-8, 1e-3, 1e-2, 1, 5, 10, 20
ridge_regressor = GridSearchCV(ridge, params, scoring = 'neg_
                    mean_squared_error', cv=5)
ridge_regressor.fit(x, y)

print(ridge_regressor.best_params_)   # return best α value
print(ridge_regressor.best_score_)   # return best mse
```

Output:

```
{'alpha': 20}
-32.380250
```

Mse for both regression,

Linear Reg — -37

Rige Reg — -32

This is best because it is low.

But, Ridge also tries to reduce Overfitting.

In this Scenario, we get linear regression because it is good.

Lasso Regression [Same Code for ridge but name change to Lasso]

Also play a n number of α params like 30, 35, 40, to give better accuracy.

We are already know cross validation splits the data and we also have test-train split.

#Code

```
from Sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split
        (x, y, test_size = 0.33, random_state = 42)
```

Then pass the code like,

```
mse = cross_val_score (lin_reg, x_train, y_train, Scoring)
```

Like this

What is difference between Cross Validation and Train Test Split?

Some points,

→ Train_Test split only splits based on random state and test_size. It gives different accuracy when Changing the random state.

→ But, Cross Validation performs Correctly based on the CV Count.

Suppose, 10k datapoints

$CV = \frac{10}{10} = 1$ [for test] 10 %

$10 - 1 = 9$ [for train] 90 %.

It gives n number of accuracy based on CV Count and take better accuracy with the help of Mean.

<u>Remember:</u>

The Performance metrics is evaluated by the test data.

MSE, MAE, RMSE → It is used for mainly to evaluate the prediction error rates. Lower MSE, MAE, RMSE the closer is to [i.e cost function = 0]. It is better fit.

R^2 and adjusted R^2:

The R^2 and adjusted R^2 is used to evaluate model performance like 80 %, 95 % and so on.

Both are regression evaluation metrics based on the usage we can use. Mostly, MSE, MAE, RMSE is used only for evaluate prediction rates. R^2 and adjusted R^2 are used to model performance.

Code:

```
# Find Accuracy of the Model
y_pred = ridge_regressor.predict (x_test)
from sklearn.metrics import r2_score
r2_score1 = r2_score (y_pred, y_test)
Print (r2_score_1).
```

Output: 0.650955

$\boxed{65\% \text{ Accuracy}}$

Logistic Regression:

Code:

```
from sklearn.linear_model import LogisticRegression

from sklearn.datasets import load_breast_cancer


df = load_breast_cancer()

# Independent feature

x = pd.DataFrame(df['data'], columns=df['features_names'])

x.head()

# Dependent feature

y = pd.DataFrame(df['target'], columns=['Target'])

y

# Check dataset is balanced or imbalanced

y['Target'].value_counts()
```

Out:

```
1    357
0    212
```

The dataset is balanced One.

Train Test Split

```
from Sklearn.model_selection import train_test_split

x_train, X_test, y_train, y_test = train_test_split
                    (x, y, test_size = 0.33, random_state = 42)


Params = [{'C':[1,5,10]}, {'max_iter' : [100,150]}]

Model1 = LogisticRegression (C=100, max_iter = 100)

Model = GridSearchCV (model1, param_grid = params, Scoring = 'f1',
                                                    cv = 5)

model.fit (x_train, y_train)


# Check best params

Model.best_params_

Out: {'max_iter' : 150}

model.best_score

Out: 0.95751136

y_pred = model.predict (x_test)

y_pred
```

Confusion Matrix

```
Confusion_matrix (y-test, y-pred)
```

Out: array ([[63 , 4],
[3, 118]], dtype = int64)

Evaluate Metrics

```
from Sklearn . metrics import Confusion_matrix, Classification_report,
                                accuracy_score

accuracy_score (y-test, y-pred)

Classification_report (y-test , y-pred) # This show all the
                                Precision, Recall and f1-Score .
```

Remember:

Max_iter default value is 100 . why set the max_iter.

Because of Some time leades to infinite loop and May be not converge.

C → Inverse of regularization strength
- ↳ weak Regularization
- ↳ Strong Regularization

Weak Regularization:

→ The weak regularization, the penalty term added lost function is relatively small.

→ It means the model's Coefficient (or) slope can take larger values, allowing to fit the training datamore closely.

→ It is suitable for large and Complex datasets and there's need to Capture intricate patterns in data.

→ However applied, The model still overfit and perform poorly on new data.

Strong Regularization:

→ The larger penalty term in the loss function.

→ The penalty discourages the model coefficients taking large values, reducing impact on prediction.

→ It is useful for small and noisy dataset and also prevent from fitting.

→ It encourages the model to focus on the most important features and Prevents Overfitting.

Naive Bayes Intution: [Classification]

This algorithm is used for Classification problem.

This algorithm is worked with the help of Baye's theorem.

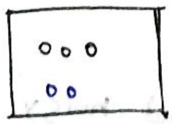
Rolling a Dice, [Independent events]

$\{1, 2, 3, 4, 5, 6\}$

$P(1) = 1/6,\ P(2) = 1/6,\ P(3) = 1/6,\ P(4) = 1/6$

So, this event is called as the independent event.

Dependent events:



o → Blue Ball

o → Black Ball

$P(\text{Black Ball}) = 3/5$

after that drawn the,

$P(\text{Blue Ball}) = 2/4 = 1/2$

This is Dependent events.

Suppose we take,

It is Somthing called Conditional probability

$$P(\text{Black and Blue}) = P(\text{Black}) * P(\text{Blue}/\text{Black})$$

So, let's write

$$P(A \text{ and } B) = P(A) * P(B/A)$$

Let's derive something,

$$P(A \text{ and } B) = P(B \text{ and } A)$$ [Both are equal, so just take this]

$$P(A) * P(B/A) = P(B) * P(A/B)$$

$$\boxed{P(B/A) = \frac{P(B) * P(A/B)}{P(A)}}$$ → Baye's theorem

So, This CRux behind Naive Bayes

How to apply in real time?

Consider,

$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad \ldots\ldots \quad x_n \qquad y$

↳ I/P feature   dependent or O/P

So, let's write

$$P(y/x_1 x_2 x_3 x_4 \ldots x_n) = \frac{P(y) * P(x_1 x_2 x_3 \ldots x_n/y)}{P(x_1 x_2 x_3 \ldots x_n)}$$

$$= \frac{P(y) * P(x_1/y_1) * P(x_2/y_2) * P(x_3/y_3) \ldots\ldots P(x_n/y_n)}{P(x_1) * P(x_2) * P(x_3) * \ldots\ldots * P(x_n)}$$

Suppose the dataset contains,

x_1	x_2	x_3	x_4	y
Independent O/P				dependent O/P
-	-	-	-	Yes
-	-	-	-	No

$$P(y = yes / x_i) = \frac{P(yes) * P(x_1/yes) * P(x_2/yes) * P(x_3/yes) * P(x_4/yes)}{\underbrace{P(x_1) * P(x_2) * P(x_3) * P(x_4)}_{\text{Fixed [constant]}}}$$

$$P(y = No / x_i) = \frac{P(No) * P(x_1/No) * P(x_2/No) * P(x_3/No) * P(x_4/No)}{\underbrace{P(x_1) * P(x_2) * P(x_3) * P(x_4)}_{\text{Fixed [constant]}}}$$

$$x_i \begin{cases} \rightarrow Yes \\ \rightarrow No \end{cases}$$

Suppose, we have

$\boxed{P(yes/x_i) = 0.13}$ $\boxed{P(No/x_i) = 0.05}$

In binary Classification, we decide

$\geq 0.5 \Rightarrow 1$ [0.5 is the threshold for

$< 0.5 \Rightarrow 0$ Comparing binary Classification]

So, let's apply normalization

$$P(Yes|x_i) = \frac{0.13}{0.13 + 0.05} = 0.72 \Rightarrow 72\%$$

$$P(No|x_i) = 1 - 0.72 = 0.28 \Rightarrow 28\%$$

This is known as the intution of Naive Bayes.

Real world Example:

Consider the DATASET,

DAY	OUTLOOK	TEMPERATURE	HUMIDITY	WIND	PLAY TENNIS
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

So, let's take Outlook feature

Outlook:

	Yes	No	P(y)	P(No)
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0/5
Rain	3	2	3/9	2/5
Total	9	5		

P(Y) → Probability of yes for given outlook values.

P(N) → Probability of No for given outlook values.

So, let's take the temperature feature

Temperature:

	Yes	No	P(y)	P(N)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cold	3	1	3/9	1/5
Total	9	5		

Then also take a PLAY feature,

↳ dependent feature.

So, Conclude that

PLAY:

Yes	9	$P(yes) = 9/14$
No	5	$P(No) = 5/14$
Total	14	

Suppose the new data will be Come,

Test
→ (sunny, Hot) ⟶ what is the O/P ?

So, let's find out

$$P(yes \mid (sunny, hot)) = \frac{P(yes) * P(sunny/yes) * P(Hot/yes)}{P(sunny) * P(Hot)}$$

$$= \frac{9/14 * 2/9 * \frac{2}{9}}{\cancel{5}/\cancel{5} * \cancel{4}/\cancel{4}}$$

[That's why Just ignore because it is fixed]

$$= \frac{\cancel{9}}{\cancel{14}_{7}} * \frac{\cancel{2}^{1}}{\cancel{9}} * \frac{2}{9}$$

$$= \frac{2}{63} \Rightarrow 0.031$$

$$P(No/Sunny, hot) = \frac{P(No) * P(sunny/No) * P(hot/No)}{P(sunny) * P(hot)}$$

↳ Just ignore [It's fixed]

$$= \frac{\cancel{5}}{\underset{7}{\cancel{14}}} * \frac{3}{\cancel{5}} * \frac{\overset{1}{\cancel{2}}}{5}$$

$$= \frac{3}{35} = 0.085.$$

So,

$$\boxed{\begin{aligned} P(yes/sunny, hot) &= 0.031 \\ P(No/Sunny, hot) &= 0.085 \end{aligned}}$$

The final thing is Normalize,

$$\text{Normalized } P(yes/Sunny, hot) = \frac{P(yes/sunny, hot)}{P(yes/sunny, hot) + P(No/sunny, hot)}$$

$$= \frac{0.031}{0.031 + 0.085}$$

$$= 0.267241 \Rightarrow 27\%$$

$$\text{Normalized } P(No/Sunny, hot) = \frac{P(No/sunny, hot)}{P(yes/sunny, hot) + P(No/sunny, hot)}$$

Just apply formula (or) Subtract whole from 1.

$= 1 - 0.267241$

$= 0.732 \Rightarrow 73\%$.

So,

$$\boxed{\begin{array}{l} P(\text{yes} / \text{sunny, hot}) = 27\% \\ P(\text{No} / \text{sunny, hot}) = 73\% \end{array}}$$

So, the threshold value is 0.5,

$P(\text{yes} / \text{sunny, hot}) < 0.5$

So, obivously take,

$$\boxed{P(\text{No} / \text{Sunny, hot}) = 73\%}$$

Therefore,

If a person play with (Sunny, hot) $\rightarrow$ O/p ?

The probability of No is high.

So, The Answer is <u>No</u>.

This is all about the Naive Bayes for binary Classification.

KNN Algorithm:

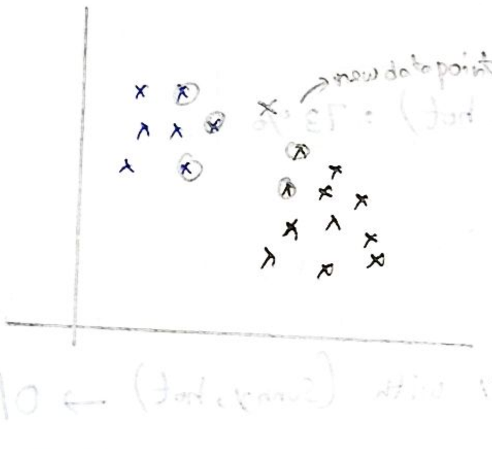
KNN Algorithm

- Classification
- Regression

→ KNN means k-nearest neighbours.

→ It also support to solve both regression and Classification problem.

Classification:



Suppose, k=5

$\boxed{k=5}$

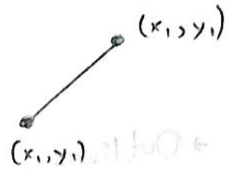
Take 5 nearest Points and Calculate the distance.

In this case, maximum point taking from blue group. So, the new point consider as a blue group.

Blue point - 3 [obiviously tis is less distance]

Black point - 2

The Calculation is done by two distance formula,

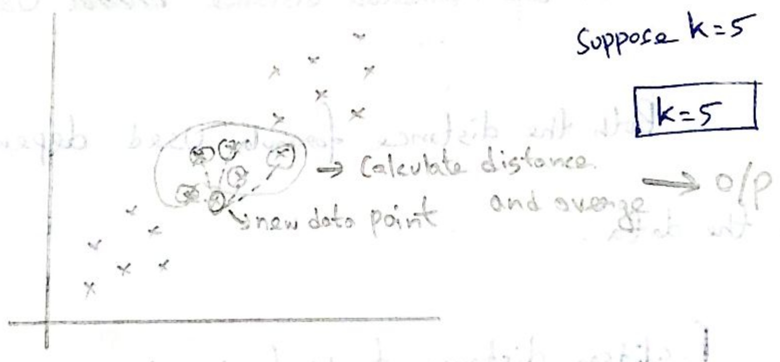
→ Eucledian Distance. 

→ Manhatten Distance. 

How Categorical feature plotted in KNN ?

Before applying KNN, The various feature encoding techniques will be used. This will be used to Convert Categorical feature into Numerical values. Then plot KNN.

<u>Regression:</u>



Take nearest five points distance and Calculate the average. This will be prediction.

k is the hyperparameter.

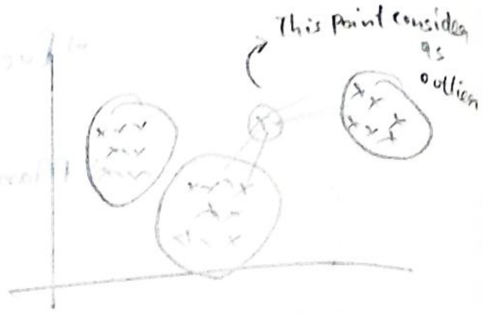
Apply the k value 1 to 50 and check the error rate.

If error rate is <u>low</u>, take this model for analysis.

K-nearest is working bad in two Cases are,

→ Outliers.

→ Imbalanced dataset.



<u>Formula,</u>

Eucledian distance $= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Manhatten distance $= |(x_2 - x_1) + (y_2 - y_1)|$ (or)

$|x_1 - x_2| + |y_1 - y_2|$

When Eucleidian and Manhattan distance ~~work~~ Use?

Both the distance formula Used depends nature of the data.

Euclidean distance tends to work well when the data is distributed in a More Circular Manner.

While, The Manhattan distance Can be More Suitable for Grid-like (or) lattice like structure.

(It means where Values are arranged in rows and Columns).