Not Sure Not Sure Yes 2

Page No.
Date

Machine learning

Feature Scaling

This is applied only to columns

* Normalization
  → $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$
  → [0;1]

* Standardization
  → $x' = \frac{x - \mu}{\sigma}$
  → [-3 ; +3]

Part 1 Data processing

Here add CSV file accordingly

1] Importing Libraries

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

2] Importing the dataset

```
dataset = pd.read_csv('Data.csv')
x = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
print(x)
print(y)
```

3] Taking care of missing data

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')
imputer.fit(x[:, 1:3])
x[:, 1:3] = imputer.transform(x[:, 1:3])
print(x)
```

4] Encoding Categorical data

Since some of data in CSV file is i.e some column has in string we gotto convert them into num so we use one hot encoding method

* Encoding the independent Variable i.e x

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransform(transformers = [('encoder', OneHotEncoder(), [0])], remainder = 'passthrough')
X = np.array(ct.fit_transform(X))
print(X)
```

(transformers: the colmn to transform; 'encoder': tell what u r doing i.e encoding; OneHotEncoder(): type of encoding i.e 1 hot encoding; [0]: index of colmn i.e column '0'; remainder: so to indicate that next or remaining colm are not need to encode)

* Encoding of Dependent Variable

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
print(y)
```

(This is used for dependent Variable)

Feature scaling should apply after the split of data just in order to avoid the leakage of any data in test set.

5] Splitting the dataset into the training set & Test Set (80% / 20%)

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 1)
print(x_train)
print(x_test)
print(y_train)
print(y_test)
```

(dummy variables: when there are created i.e 80% train & 20% test for both x & y these data are called dummy variable)

Page No.
Date

Exercise

```
#import library
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
                                  ↓ This is used for feature scaling
#load iris.csv
iris_df = pd.read_csv('iris.csv')
#Seperate features & target
X = iris_df.drop('target', axis=1)
y = iris_df['target']

#Split data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#To apply feature scaling
Scaler = StandardScaler()
X_train = Scaler.fit_transform(X_train)
X_test = Scaler.transform(X_test)
```

Regression

Simple linear regression $y = b_0 + b_1 x_1$

Multiple linear regression $y = b_0 + b_1 x_1 + b_2 x_2 + \dots + b_n x_n$

Polynomial linear regression $y = b_0 + b_1 x_1 + b_2 x_1^2 + b_3 x_1^3 + \dots b_n x_1^n$

Page No.
Date

↳ It is not reqd to apply on *) Column that have binary Value
*) On dummy Variable i.e result of One hot encoder

6] Feature Scaling

```
from sklearn.preprocessing import StandardScaler
SC = StandardScaler()
X_train[:, 3:] = SC.fit_transform(X_train[:, 3:])
X_test[:, 3:] = SC.transform(X_test[:, 3:])
```

it will just find fit & transform
it will apply Normalization / Standardisation

↳ for testing we only have to use transform method & no fit method.

→ Feature Scaling is applied only to the main numerical data Ex: while encoding we have converted string into int value, these generated int are dummy Variable hence feature scaling is not applied here.

10:31 AM
30/4/24

* Here dependent var Y has binary value so feature scaling is ain't applied

Part 2

Regression

List of Regression models :-

1. Simple Linear Regression
2. Multiple Linear Regression
3. Polynomial Regression

} feature scaling is not reqd
Explisit i.e y depends on the RHS

4. Support Vector for Regression → Implicit i.e y doesn't only depend on RHS
5. Decision Tree Regression
6. Random Forest Regression

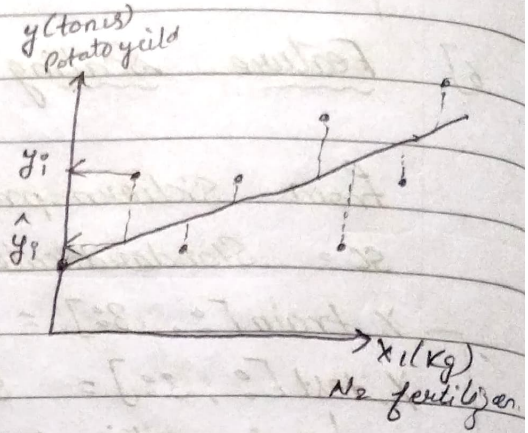
1. Simple Linear Regression

$$\hat{y} = b_0 + b_1 X_1$$

- $\hat{y}$: Dependent Variable
- b_0 : y intercept (Const)
- b_1 : Slope Coeff
- X_1 : independent Variable

Page No.
Date

Ordinary least Sq method



residual = $\varepsilon_i = y_i - \hat{y}_i$

$\hat{y} = b_0 + b_1 x_1$

$\text{Sum}(y_i - \hat{y}_i)^2$ should be minimum

Steps

1] Importing Libraries
2] Importing Dataset → salary_Data.csv
3] Splitting data into training & test Set
4]

4] Training the Simple Linear Regression model on the Training Set

From sklearn.linear_model import LinearRegression
regressor = LinearRegression() → build a model
regressor.fit(X_train, y_train) // to fit is used to train regression model

5] Predicting the Test Set result

~~regressor~~
y_pred = regressor.predict(X_test)

6] Visualizing the Training Set results
→ graph on x_train & y_train values

graph which predict salary using available x_train value. This is generally regression line

```
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary Vs experience (Training Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

60
23

Page No.
Date

Visualising the test set result

Here we didn't use X_test parameter bcz we will get same regression line for both

```
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary Vs experience (Test Set)')
plt.xlabel('Years of experience')
plt.ylabel('Salary')
plt.show()
```

Additional Exercise

* How to make a single prediction?, like what Salary would be the salary of employee with 12yrs of experience?

→
```
print(regressor.predict([[12]]))
```

12 → Scalar
[12] → 1D
[[12]] → 2D

* How to get final linear regression eq^ with values of Co-eff?

→
```
print(regressor.coef_)
print(regressor.intercept_)
```

ans → [9345.942...]
26816.1922...

∴ final eq^ will be

Salary = 26816.19 + 9345.94 × Years Experience

$\hat{y} = b_0 + b_1 x_1$

Page No.
Date

Here no need of feature scaling

2. Multiple Linear Regression

Dummy Variable

dependent — Independent — Dummy Variable

Profit	R&D Spend	Admin	Marketing	State	N-Y	California
192261.83	165349.20	136897.80	471784.10	N-Y	1	0
191792.06	162597.70	151377.59	443898.53	California	0	1

(X1) (X)

$y = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + b_4D_1$ → Here D considered only N.Y bcz whereas D1 is '0' is dipicted California

Always Vomit one dummy Variable (if there are 90 dummy Variable then include 98 i.e Vomit 1)

Statistical Significance

Tail	0.5	So when a Coin is Tossed but you continuously get Tails in row, so at 1 pt it becomes Suspecious that Smtg must be wrong with Coin or the Coin may not be fair. Generally we consider 95% as Statistical Significance reference.
Tail	0.25	
Tail	0.12	
Tail	0.6 — 0.5	
Tail	0.3	
Tail	0.7	

Building a Model

5 methods of building a Model

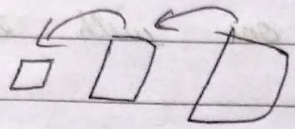
1. All-in
2. Backward Elimination
3. Forward Selection
4. Bidirectional elimination
5. Score Comparison

} Stepwise regression

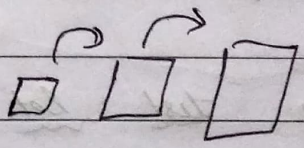
Page No.
Date

1 All-in

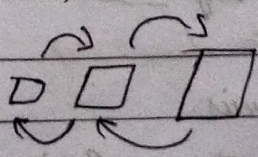
* Prior knowledge ; OR
* You have to; OR
* Preparing for backward elimination

2 Backward Elimination 

Step1 : Set Significance level (SL = 0.05 i.e 5%.)
Step2 : Fit the full model with all possible predictors
Step3 : Consider the predictor with the highest P.value.
If P > S.L . go to Step 4 otherwise to FIN
Step4 : Remove the predictor
Step5 : Fit model without this Variable

3 Forward Selection 

Step1 : Set Significance level eg SL=0.05
Step2 : Fit all Simple regression models $y \sim x_n$ Select the one with the lowest P-Value
Step3 : Keep this Variable & fit all possible models with one extra predictor added to the one(s) you already have
Step4 : Consider the predictor with the lowest P-value.
If P<SL go to Step3, Otherwise go to FIN

4 Bidirectional Elimination 

Step1 : Sel S.L i.e SLENTER = 0.05 & SLSTAY = 0.05
Step2 : Perform next step of forward Selection (new var must have P<SLENTER to enter)
Step3 : Perform all step of Backward elimination (old var must have P<SLSTAY to stay)
Step4 : No new Variable can enter & no old var can exit

5 All possible Models

Step1 : Select a Criteria of goodness of fit (eg Akaike Criterion)
Step2 : Construct all possible regression models : $2^N - 1$ Comb
Step3 : Select one with the best Criterion

Steps of multiple Linear regression

1. Importing libraries
2. Import dataset → 50_startups.csv — R&D, Admin, State, Profit
3. encoding the data — Here we are encoding the data V as we are training the data include all the categorical data which is encoded into 0 & 1
4. Splitting data into train & test set
5. Training multiple linear regression model on the train set

6 Predicting the test set result

```
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2) → //only 2 digit after decimal
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape
(len(y_test),1)), 1))
```

reshaping is done so that it will print vertically
1 col/m
1 specify axis=1 i.e horizontal (0 for vertical & 1 for horizontal)

//Syntax for print concatenate is: Concatenate((a1, a2, ...), axis=0, out=None)
here 2 vari i.e a_1 & a_2 are y_pred & y_test

9:20AM
2/5/2024 3] Polynomial Regression → Here no need of splitting of data bcz we req whole data

1 Importing Libraries
2 Import dataset → Position_Salaries.csv
3

optional
3 Training the Linear regression model on the whole dataset

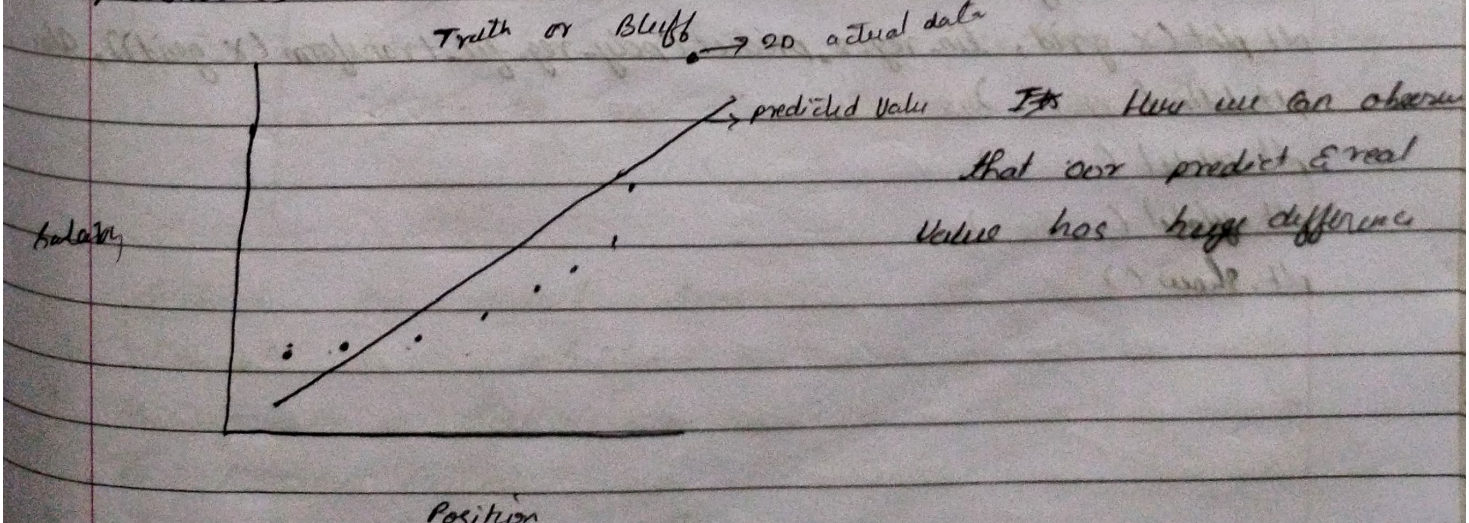
```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
```

4 Training the polynomial Regression model on whole dataset

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 2)   → gives it mean n=2
X_poly = poly_reg.fit_transform(X)   → Here we created new matrix of features upto n=2 value i.e y = b0 + b1x1 + b2x1^2
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

5 Visualizing the Linear Regression result

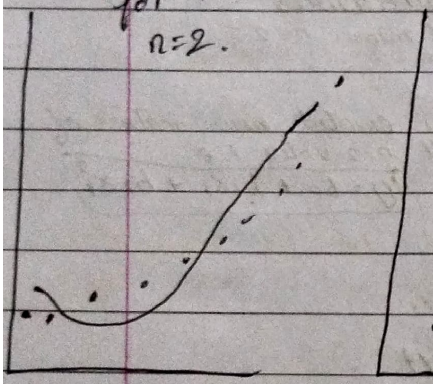
```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Truth or Bluff (Linear regr)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



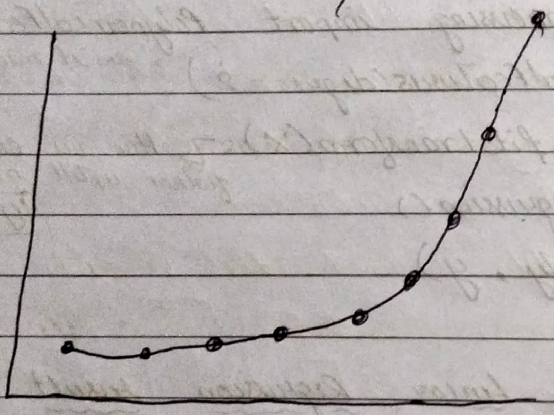
6 Visualising the polynomial Regrn Result

```
plt.scatter(X, y, color='red')          → or poly_reg.fit_transform(x)
plt.plot(X, lin_reg_2.predict(X_poly), color='blue')
plt.title(  . . . . )
plt.xlabel( . . . . )
plt.ylabel( . . . . )
plt.show()
```

for n=2.



This is for n=4.



Optional

7 Visualizing the P-R result (for higher resol^n & smoth curve)

```
X_grid = np.arrange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color='red')
plt.plot(X_grid, lin_reg_2.predict(poly_reg.fit_transform(X_grid)), color='blue')
plt.title(  )
plt.xlabel(  )
plt.ylabel(  )
plt.show()
```

8 Predicting a new result with linear Regression

row col

lin_reg.predict([[6.5]])

→ array([[330378.78]])

→ ₹

9 Predicting a new result with Polynomial Regression

lin_reg_2.predict(poly_reg.fit_transform([[6.5]])

→ array([158862.45])

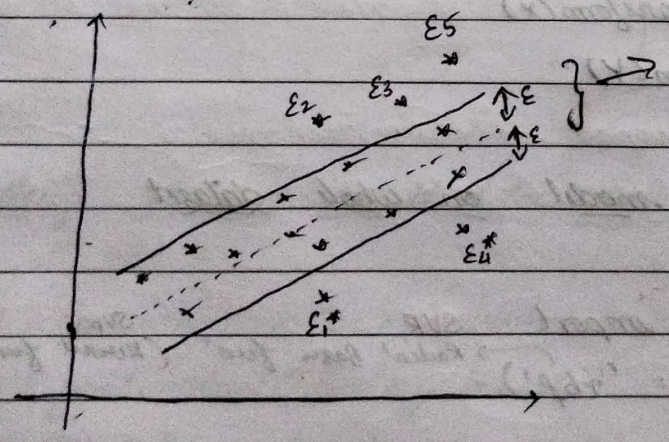
For levels 6 Salary was 150K

For level 7 " " 200K

→ ∴ predicted Salary for 6.5 level should be in rang ~~150K~~ 150k ≤ Salary ≤ 200k but linear regression predicted 330K ∴ its not recommended for curve data (parabolic)

This is

4 Support Vector Regression (SVR Intuition)



ε - Insensitive Tube

slack variable above ε* below ε*

ε1 to ε5 are supporting the tube

Sections on Kernel SVM

* Kernal SVM intuition
* Mapping to a higher dimension
* The Kernel Trick
* Types of Kernel fun^n
* Non-linear Kernel SVR

Position	Level	Salary
Business Analyst	1	45K
⋮	⋮	⋮
CEO	10	1M

4:45AM

3/5/2024 Steps

1 Importing Libraries

2 Importing the dataset → 'Position_Salaries.csv'

Here while import data & assign X & Y vari

3 ~~Feature Scaling~~ resp for feature & depend var

at that time the dependent var is in 1D but in order to scale it StdScaler expects the i/p in 2D only ∴ we have to edit the Y before scaling it otherwise it will throw an error.

∴ y = y.reshape(len(x), 1)

3 Feature scaling

```
From sklearn.preprocessing import StandardScaler
SC_X = StandardScaler()   } Here 2 diff objects are created by StdScaler will genrate
SC_Y = StandardScaler()   } M & C but out X & Y have huge difference hence we have to create 2diff instances
X = SC_X.fit_transform(X)
Y = SC_Y.fit_transform(Y)
```

4 Training the SVR model on whole dataset

```
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')   → Radial Basis fun"   (SVM Kernel fun")
regressor.fit(X, y)
```

Now we have to predict the ~~Sandy~~ Salary for 6.5 level. But earlier we have scaler ~~6.5~~ X & Y ∴ we have to transform this inversely & get the reqd result.

→

4946

Page No.
Date

5 Predicting new Result

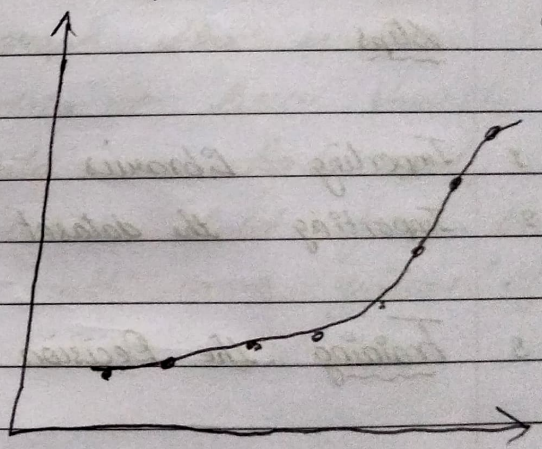
```
sc_y.inverse_transform(regressor.predict(sc_x.transform([[6.5]]))
                                .reshape(-1,1))
```

→ array([[170370.02]])

↳ this is just to get rid of error

6 Visualizing the SVR Result

```
plt.scatter(sc_x.inverse_transform(x), sc_y.inverse_transform(y)
                                , color='red')
plt.plot(sc_x.inverse_transform(x), sc_y.inverse_transform(regressor
                .predict(X).reshape(-1,1)), color='blue')
plt.title('Truth or Bluff (SVR)')
plt.xlabel('Position Level')
plt.ylabel('Salaries')
plt.show()
```



7 Visualizing the SVR result (for high reso^n & smooth curve)

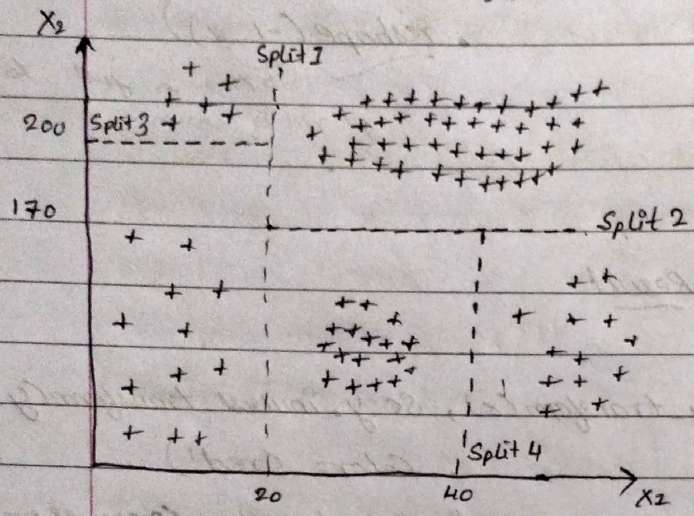
```
X_grid = np.arange(min(sc_x.inverse_transform(X)), max(sc_x.inverse_transform
                                (X)), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(sc_x.inverse_transform(X), sc_y.inverse_transform(y), color='red')
plt.plot(X_grid, sc_y.inverse_transform(regressor.predict(sc_x.transform(X_grid))
                .reshape(-1,1)), color='blue')
plt.title(' . . . ')
plt.xlabel(' . . . ')
plt.ylabel(' . . . ')
plt.show()
```

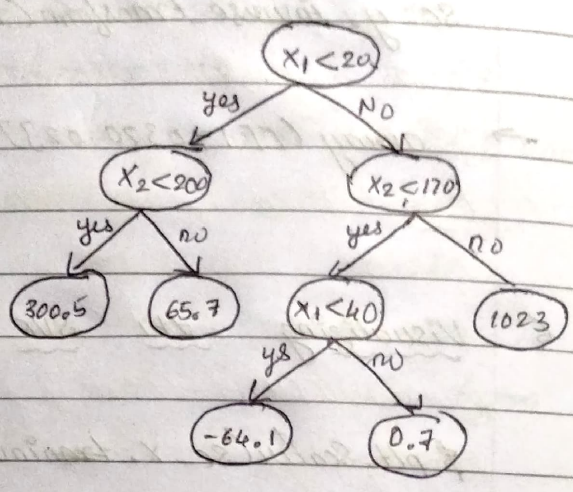
CART → classification and Regression Trees

5 Decision Tree Regression

no feature scaling

Decision Tree Intuition





Steps

1. Importing Libraries
2. Importing the dataset
3. Training the Decision Tree Regression model on the whole dataset

```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X, y)
```

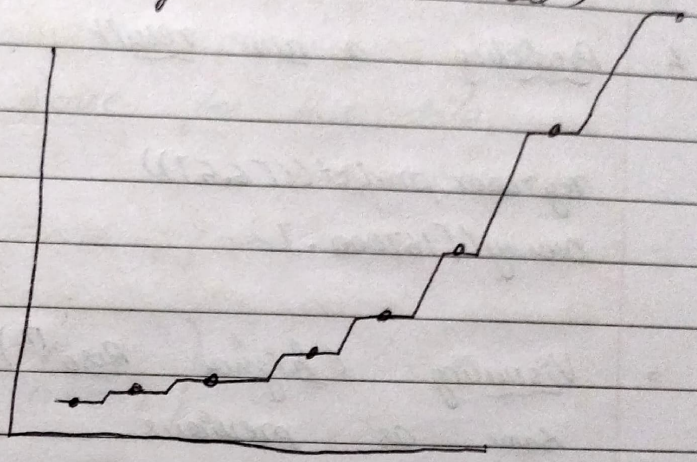
4. Predicting new Result

```
regressor.predict([[6.5]])
array([150K])
```

Page No.
Date

5 Visualizing the Decision Tree Regression result (High Resol")

```
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
plt.title (' ... ')
plt.xlabel (' ... ')
plt.ylabel (' ... ')
plt.show()
```



We didn't get proper o/p or graph, this is bcz decision tree is not fit less data i.e lesser ~~Variable~~ features. Here in the given data we had only 1 feature i.e level so it didn't hold good. But we can use this for data with more features # of matrix

6 Random Forest Regression

Steps

1 import libraries
2 import dataset

→

3 Training the Random Forest Regression model on whole data

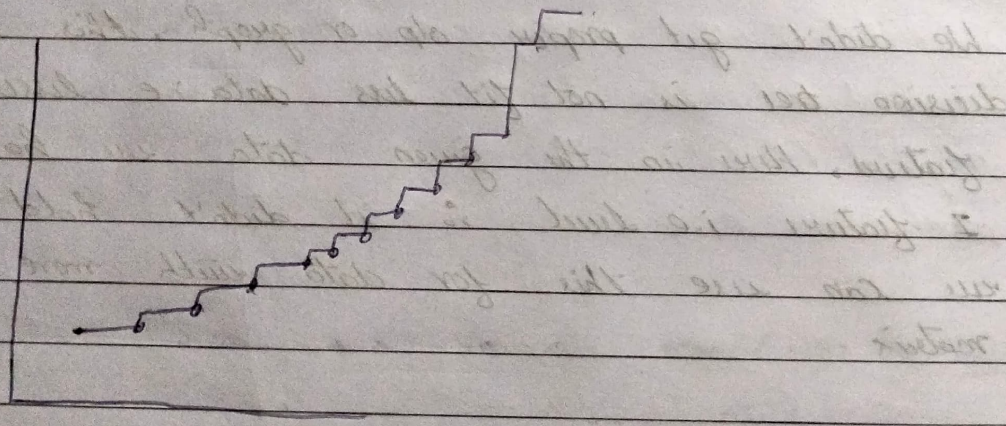
```
From sklearn.ensemble import RandomForestRegression
regressor = RandomForestRegression(n_estimators = 10, random_state = 0)
regressor.fit(X, y)
```
↓ no. of trees

4 Predicting a new result

```
regressor.predict([[6.5]])
array([167000.])
```

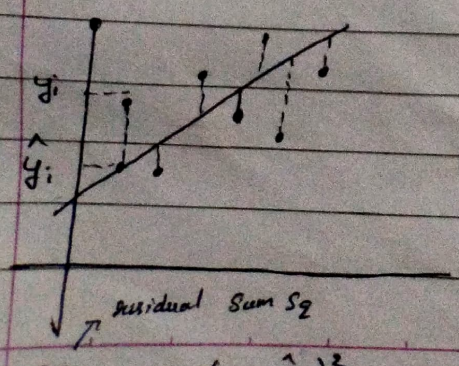
5 Visualing (higher Reso^n)

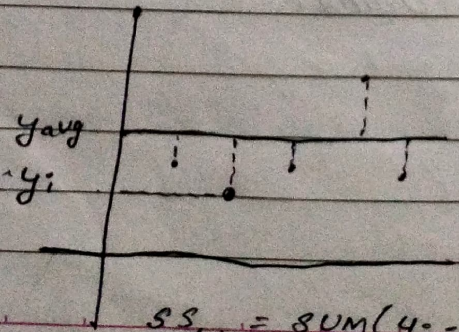
Same as previous



Even this method is not fit for 2 dataset with less features

R Squared





Page No.
Date

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$ → It lies btwn 0 & 1.

Rule of Thumb

1.0 = Perfect fit (Suspicious)

0.9 = Very good

< 0.7 = Not great

< 0.4 = Terrible

< 0 = Model makes no sense for this data.

Adjusted R Square

$$Adj\ R^2 = 1 - (1 - R^2) \times \frac{n-1}{n-k-1}$$

k = num of independent Variables

n = Sample Size

Regression Model in Python.

A-Z model Selection ↑ Template

Here we gotta make use of it & evaluate the efficiency for all.

To evaluate model performance we have,

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

↳ This predicted Value should be closer to 1

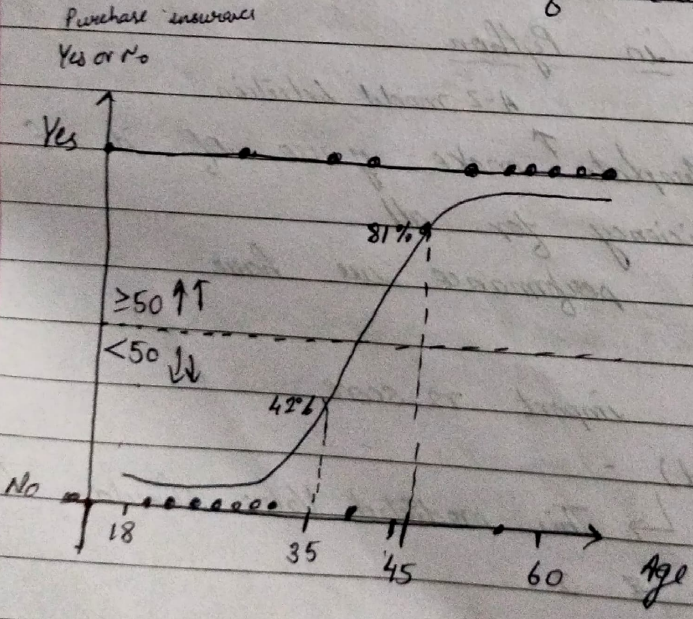
Part 3 - Classification

Classification models

i Logistic Regression
ii K-Nearest Neighbours (K-NN)
iii Support Vector Machine (SVM)
iv Kernel SVM
v Naive Bayes
vi Decision Tree classification
vii Random Forest classification

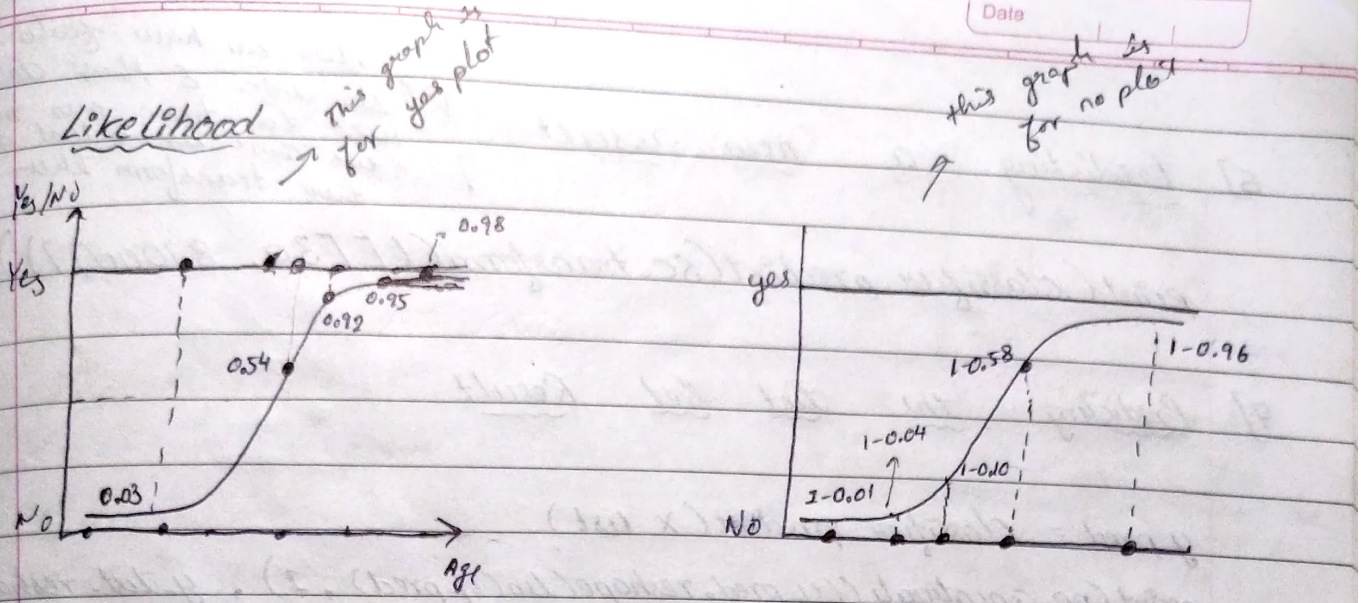
i] Logistic Regression

It predict a categorical dependent Variable from a num of indipendent Variables.



$$\frac{\ln p}{1-p} = b_0 + b_1 x_1 + \cdots b_n x_n$$

Likelihood



Likilihood = 0.03 X .54 X .92 X .95 X .98 X (1-.01) X (1-0.04) X (1-.10) X (1-.58) (1-.96)

This way likilihood is found out & Curve with max likilihood is Selected

i] Logistic Regression ⇒ Its a linear classifier

Steps

1] Importing Library
2] Importing Dataset
3] Splitting
4] Feature Scaling

5] Training the logistic Regression model on Training Set

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

6] Predicting a new result

Now we have feature Scaling the data & those data we used to train our model ∴ we can't use actual data here we transform them

```
print(classifier.predict(sc.transform([[30, 87000]])))
```

7] Predicting the test set Result

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred), 1), y_test.reshape
                                    (len(y_test), 1)), 1))
```

8] Making Confusion Matrix → To detect num of correct & incorrect predictions

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

→ { [0 1]

{ [[65 3]
[8 24]]

{ 0.89

65 correct pred of Customer who didn't buy SUV Car

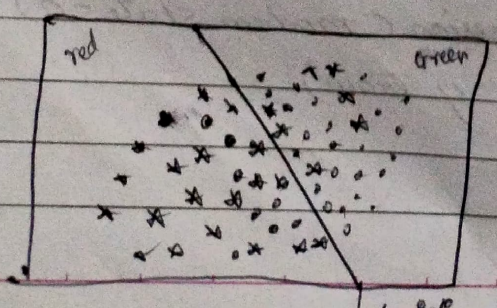
24 correct pred of Customer who did buy SUV Car

3 incorrect pred of class 1 i.e our model predicted Car didn't purchased but actually the Car was purchased

8 incorrect pred of class 0 i.e 8 Customers didn't buy ~~test~~ but we predicted buyed

9] Visualizing Training Set
10] " test set

} This is not requir but you can copy that code from CO.

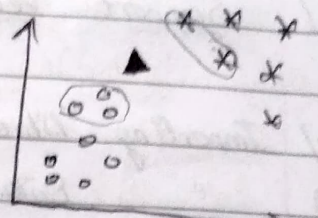


↓ Logistic regression is linear classifier

1 0 -1

Page No.
Date

ii] K-Nearest Neighbours (K-NN)



Δ belongs to ○ or ✱?
find Δ nearest neighbour then count total num of nearest no.
In this Δ has 3 ○ nearest & 2 ✱ ✱ nearest ∴ ○ wins

Steps

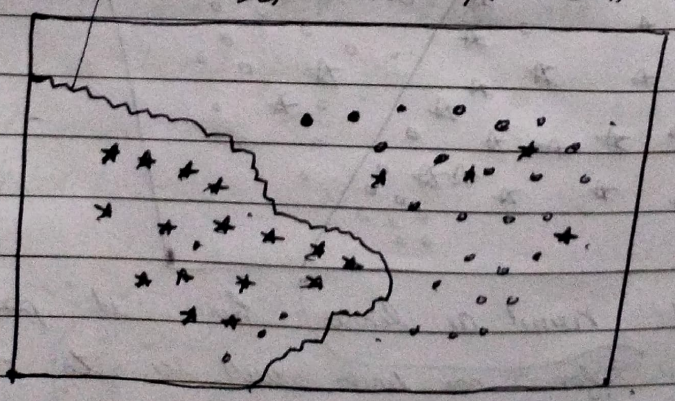
1 Importing Libraries
2 " Dataset
3 Splitting
4 Feature Scaling
5
5 Training the K-NN model on training set

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

default value 5 → no. of neighbours

This 2 are for ~~Euclid~~ Euclidean distance

6 Predicting new result
7 Predicting & Test Set result
8 Making Confusion matrix → [[64, 4] [3, 29]] 0.93
9 Visualizing training Set result
10 " Test " "

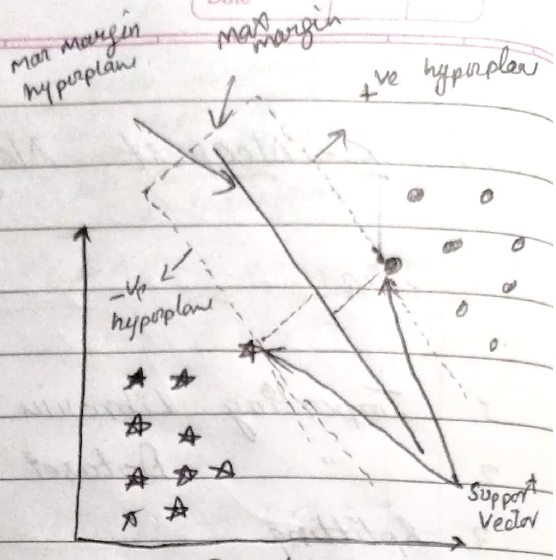


Here this line is non-linear & hence it covers major pt that belong to particular region hence its accuracy prediction is far better than logistic regression

It is non linear model

Page No.
Date

iii] Support Vector machine



line is drawn equidistant from Support Vector i.e Support Vector acts as reference pt

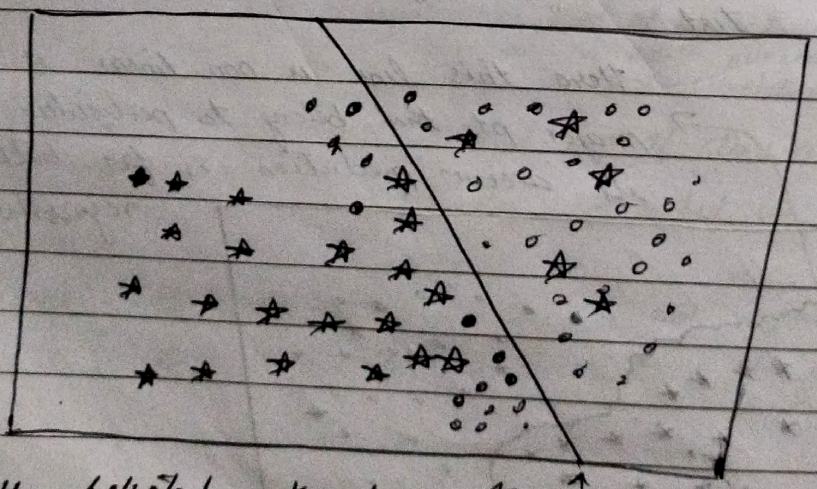
1] Importing Libraries
2] " Datasets
3] Splitting
4] Scaling

5] Training SVM on training set

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```

→ Here we chose kernel as linear we can choose anything we wanted

6] Predicting one result
7] " Test set
8] Confusion matrix → `[[66 2]` `[8 24]]` 0.9
9] Visualizing training set
10] " " Test set



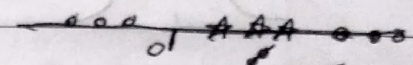
As we selected kernel as linear hence it draws straight line for non linear we will see in next concept i.e Kernel SVM

iv] Kernel SVM

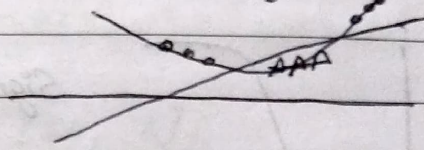
Here we can treat linearly inseparable data

Ex:

$\rightarrow f = x-5$

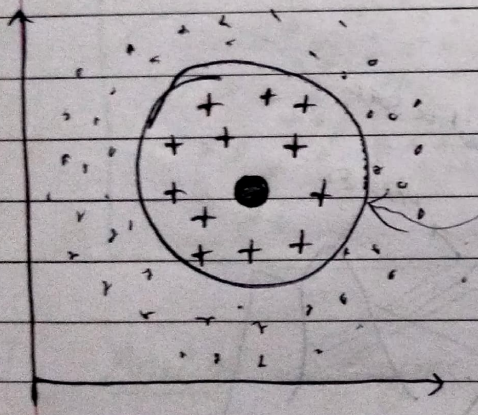


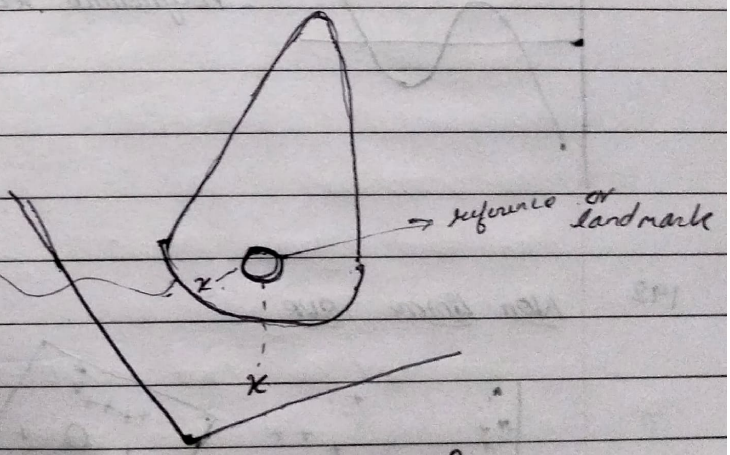
Now $f = (x-5)^2 \rightarrow$



This way we can separate linearly inseparable data

The Gaussian RBF Kernel





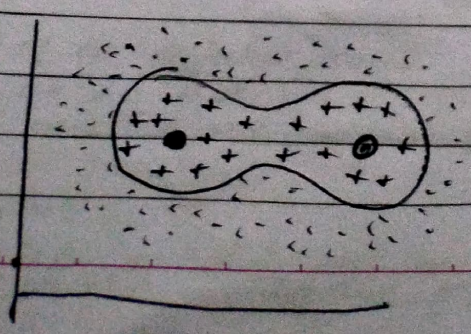
$$K(\vec{x}, \vec{l}^i) = e^{-\frac{\|\vec{x}-\vec{l}^i\|^2}{2\sigma^2}}$$

l = is the dist from center

when $\vec{x}-\vec{l}^i$ tells abt the dist

If x is located far from refurince then distance increase, & of it make even more $\therefore e^{-\text{large num}}$ would result the value closer to zero

If



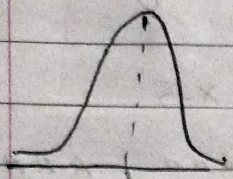
$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2)$

for + : $K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2) > 0$

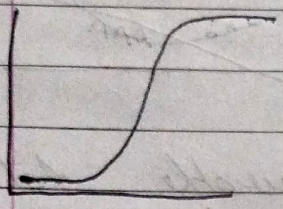
for • : " " $= 0$

Page No.
Date

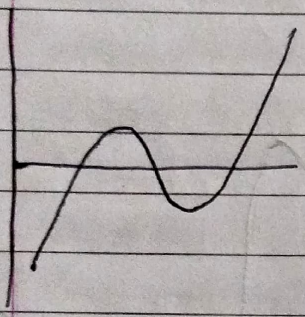
Types of Kernel func



Gaussian RBF Kernel $K(\vec{x}, \vec{l^i}) = e^{-\frac{\|\vec{x} - \vec{l^i}\|^2}{2\sigma^2}}$

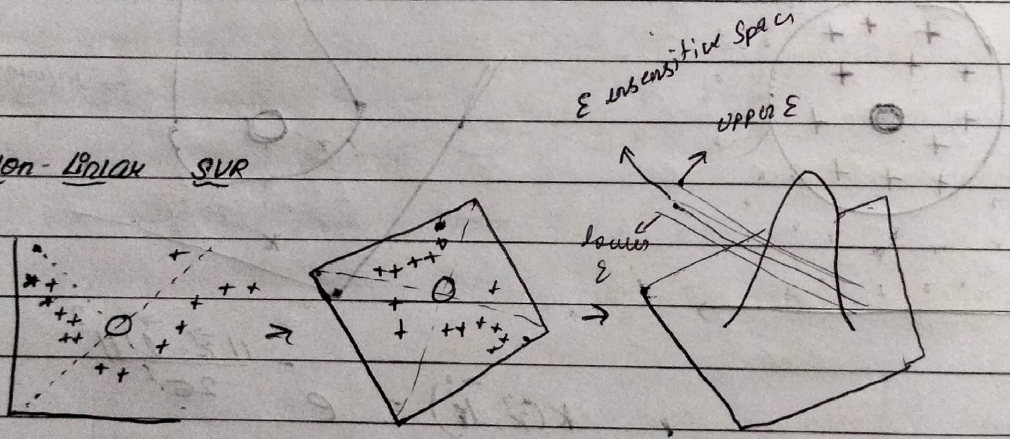


Sigmoid kernel $k(x,y) = \tanh(y \cdot x^T y + r)$



Polynomial kernel $k(x,y) = (y \cdot x^T y + r)^d, y > 0$

192 Non-Linear SVR



Steps of Kernel SVM

1. Importing Libraries
2. " Dataset
3. Splitting
4. Feature Scaling

→

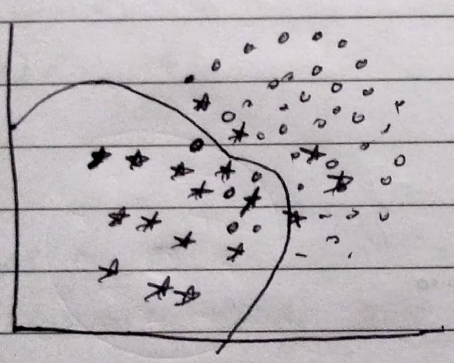
11/12.

Page No.
Date

5 Training

```
from sklearn.svm import SVC
classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(X_train, y_train)
```

6 Predicting new result
7 " Test "
8 Making Confusion matrix → [[64, 4]
[3, 29]]
0.93
9 Visualizing training set
10 " test "



[V] Naive Bayes

198

Bayes Thiorem

Mach 1 : 30 wrenchis/hr. P(mach1) = 30/50 → 0.6
Mach 2 : 20 " " P(mach2) = 20/50 → 0.4

out of all produced parts:
we can see that 1% are deffective P(diffective) = 1%

Out of the diffective parts:
we can see that 50% came from mach1 & 50% from mach2 P(mach2/Defect) = 50%.
P(mach2/Defect) = 50%.

Question?
what is the prob that part coming from mach2 is diffective = ? P(Defect/Mach2) = ?

Page No.
Date

∴ apply Bayes Theorem

$$P(A/B) = \frac{P(A)}{P(B)} \times P\left(\frac{B}{A}\right)$$

$$\therefore P(\text{Defect} / \text{mach2}) = \frac{P(\text{Defect})}{P(\text{mach2})} \times P(\text{mach2}/\text{Defect})$$

$$= \frac{1\%}{40\%} \times 50\%$$

$$= \frac{0.01 \times 0.5}{0.4} = 0.0125 = 1.25\%$$

Example

1000 wrenches

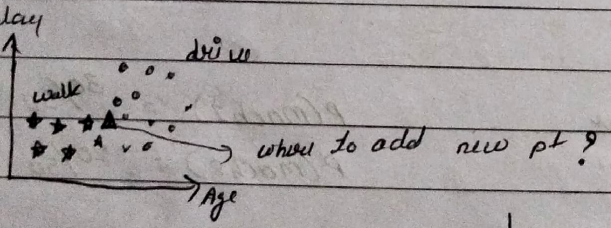
400 Cami from mach2 (1000 → 2% = 10)

1% have defect = 10

Of them 50% Cami from mach2 = 5

% of defect parts from mach2 = 5/400 = 1.25%

Q.?



∴ we have 2 steps (likelihood, prior priority)

1] $P(\text{walks}/x) = \frac{P(x/\text{walks}) * P(\text{walks})}{P(x)}$ → Posterior, → marginal

2] $P(\text{Drive}/x) = \frac{P(x/\text{drives}) * P(\text{drives})}{P(x)}$

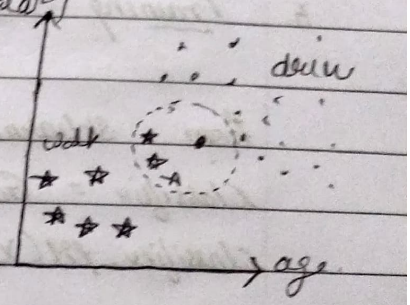
3] Compare P(walk) Vs P(Drive/x)

Step 2

* Step 2 : $P(walks) = \frac{\text{No. of walkers}}{\text{tot observation}} = \frac{10}{30}$.

$P(walks) = \frac{10}{30}$

* Step 2 : $P(x)$: To find $P(x)$ draw ing circle of ur choice radius. Consider all pt falling under this circle are similar to the pt that we are predicting



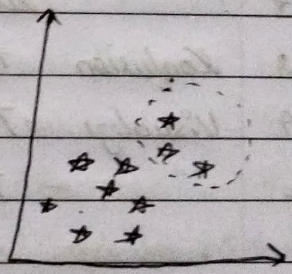
$P(x) = \frac{\text{No. of similar observations}}{\text{Tot observation}}$

$P(x) = \frac{4}{30}$

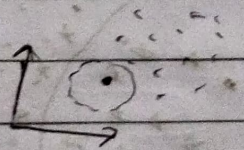
* Step 3 : $P(x|walks)$ =

$P(x|walks) = \frac{\text{Among those who walks}}{\text{Tot num of walkers}}$

$= \frac{3}{10}$.



$\therefore P(walks|x) = \frac{\frac{3}{10} \times \frac{10}{30}}{\frac{4}{30}} = 0.75$.



Step 2 : $P(drive|x) = ?$

$P(drive) = \frac{20}{30}$, $P(x) = \frac{4}{30}$, $P(x|drives) = \frac{1}{20}$

$P(drives|x) = \frac{\frac{1}{20} \times \frac{20}{30}}{\frac{4}{30}} = 0.25$

Step 3 Compare

$P(walks|x) > P(drives|x)$

$0.75 > 0.25$

Result : $\therefore$ The new pt belong to the catagory of walk

209

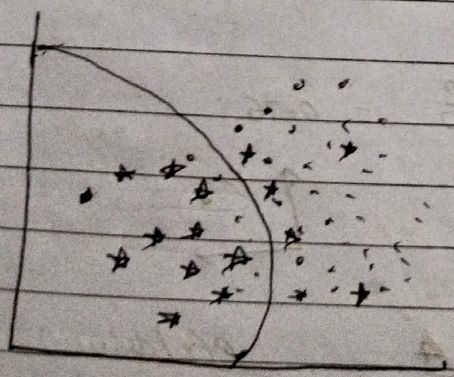
* Steps of Naive Bayes

1 Importing Libraries
2 Dataset
3 ~~Feature~~ Splitting
4 Feature Scaling

5 Training

```
from sklearn.naive_bayes import GaussianNB
Classifier = GaussianNB()
Classifier.fit(X_train, y_train)
```

6 Predicting new result
7 " test "
8 Confusion matrix → [[65, 3]
[7, 25]]
⇒ 0.9
9 Visualize Training
10 " Test

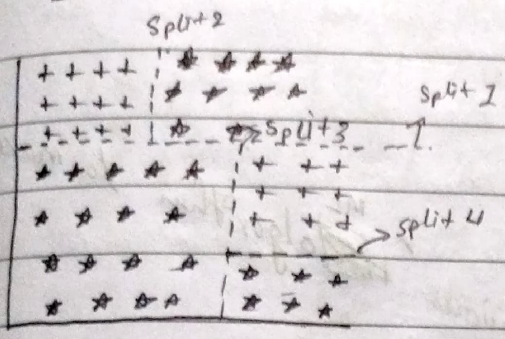


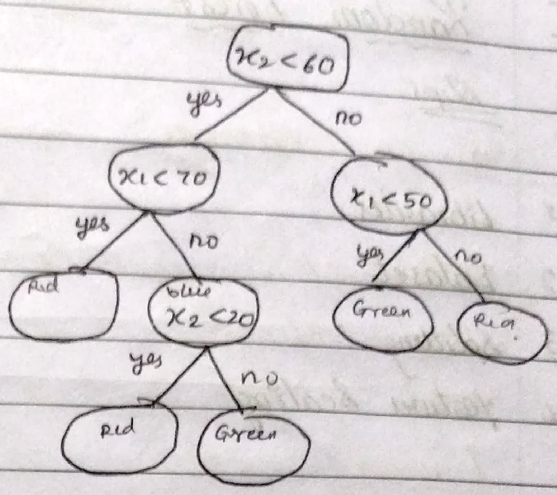
Page No.
Date

VI Decision Tree

Intuition

★ = Red
+ = Green





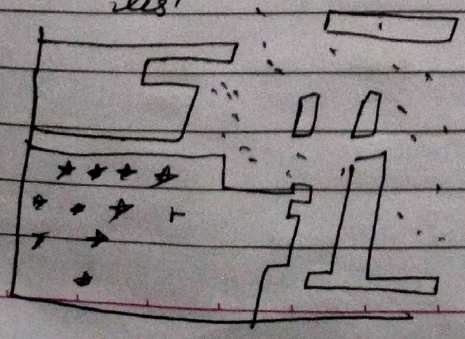
Steps

1. Libraries
2. Dataset
3. Split
4. feature Scaling

3. Training

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

6. Predict new result
7. " Test "
8. Confusion matrix → [[60 6] [3 29]] → 0.91%
9. Visualizing training
10. " Test



V Random Forest

Steps

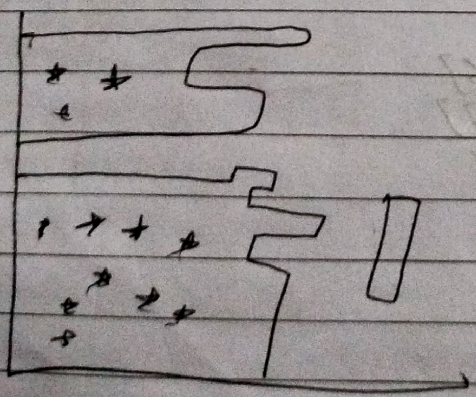
1 Libraries
2 Dataset
3 Splitting
4 Feature Scaling
6 Training

Grouping of various ML algo. to make a ML algorithm

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy'
                                    , random_state = 0)
classifier.fit(X_train, y_train)
```

6 Predict new result
7 " test " → [[63, 5]
[4, 28]]
8 Confusion matrix → 0.91
9 Visualize training set
10 " test "

20



Confusion Matrix & Accuracy

		Prediction	
		Negative	Positive
Actual	Negative	True Neg	False Pos
	Positive	False Neg	True Posit

⇓ ie

43	12
4	41

Accuracy & Error Rate

$$AR = \frac{\text{Correct}}{\text{Total}} = \frac{TN + TP}{\text{Total}} = \frac{84}{100} = 84\%$$

$$ER = \frac{\text{Incorrect}}{\text{Total}} = \frac{FP + FN}{\text{Total}} = \frac{16}{100} = 16\%$$

TN : Predicted -ve & Actual -ve

TP : " +ve & " +ve

FP : " +ve & " -ve

FN : " -ve & " +ve

Accuracy Paradox

		$\hat{y}$ (Predicted DV)	
		0	1
y (Actual DV)	0	9700	150
	1	50	100

$$AR = \frac{9800}{10000} = 98\%$$

ER :

		$\hat{y}$ (Predicted DV)	
		0	1
y (Actual DV)	0	9850	0
	1	150	0

$$AR = \frac{9850}{10000} = 98.5\%$$

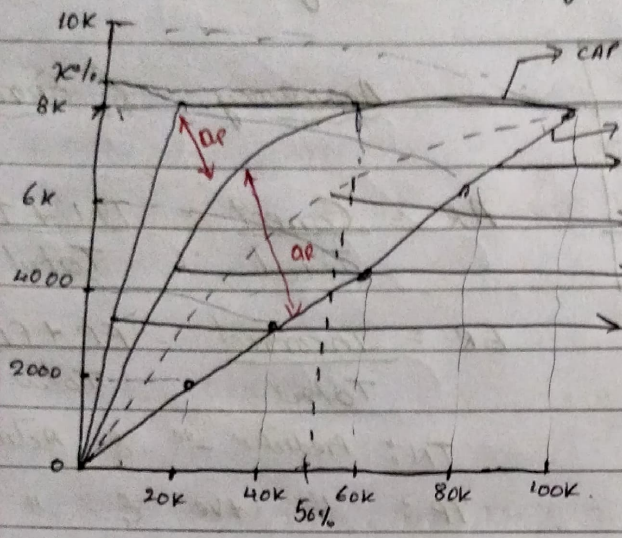
∴ AR ↑ by 5%. hence we can't just depend on AR only. this is called Accuracy Paradox.

CAP

⇒

ROC : Receiver Operating Characteristic

228 ★ CAP (Cumulative Accuracy Profile) Curve



∴ Larger the dist b/w CAP curve & A more than the model is good & reliable

227 Δ ★ AR = $\frac{aR}{aP}$ } But this is difficult to analysis Visually

★ ∴ Draw a line from 50% & follow this rule ∴

★ 90% < x < 100%. Too good

★ 70% < x < 90%. Very good

★ 60% < x < 70%. Poor

Δ ★ X < 60%. Rubbish

Part 3

232 ★ Clustering : Grouping unlabelled data

★ Supervised Learning : Here the i/p data has all the feature & model can train properly based on i/p

↓ Eg: Regression

Classification → Eg: if u train model with apple & between you can give another fruit now it can predict whether its apple or not

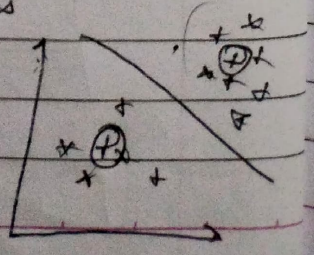
★ Unsupervised Learning : Here we dont have proper answers

Eg: Clustering

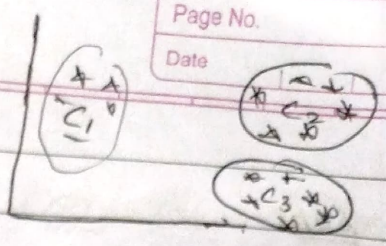
★ Eg: Here different fruits are given as i/p to train the model will grp that fruits based on similarity

233 ★ A] k-means clustering → Elbow method used to find no. of clusters

mark centroid, mark equidistance, group the pt, mark the centre of mass, replace the centroid & repeat the step



Page No.
Date



The Elbow Method

$$WCSS = \sum_{P_i \text{ in cluster } 1} dist(P_i, C_1)^2 + \sum_{P_i \text{ in cluster } 2} dist(P_i, C_2)^2 + \sum_{P_i \text{ in cluster } 3} dist(P_i, C_3)^2$$

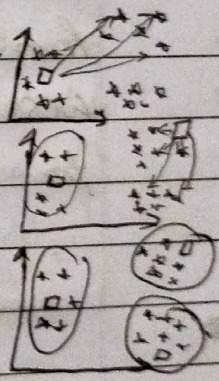
WCSS → within cluster Sum of Sq

↑ Sq of dist from centroid & each pt belong to particular cluster.

* Max no. of cluster = Tot num of pt
* As no. of cluster ↑ WCSS ↓

K-Means++

Random Installation trap: When we select incorrect or random centroid at the time of installation then no. of clusters may vary or result may be incorrect hence So address this we have K-Means++ method.



* Here first 1 random centroid is selected
* From that centroid dist of all datapt is measured
* The longest dist is chosen as nxt centroid
* The above steps are replated

~~Steps~~

Here we have data that will not have dependent variable

Steps

1 Importing Libraries

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

2 Importing dataset

```
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[ : , [3,4] ].values
```

→ Here we don't have y variable, ↑ here only 3 & 4 nic is involved so we can have 2D graph

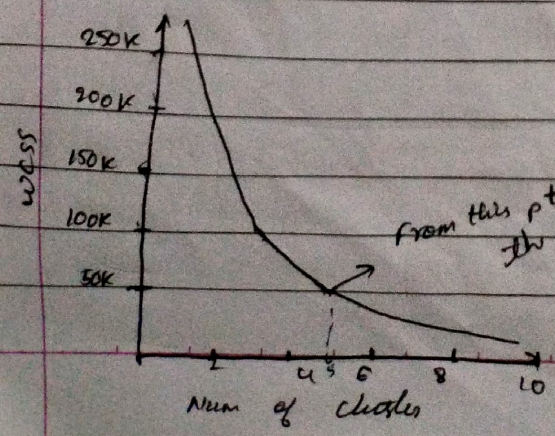
241* 3 Using the elbow method to find optimal num of clusters

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Num of clusters')
plt.ylabel('WCSS')
plt.show()
```

bcz to avoid random initialization trap

↑

→ This is the attribute



From this pt the graph rapidly decreases hence the optimal clusters = 5 ∴ optimal no. of clusters = 5

4 Training the k-Means model on the dataset

```
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42)
y_kmeans = kmeans.fit_predict(X)
print(y_kmeans)
```

→ fit_predict(X) → This will train as well as find the dependent var depending on this similarity

→ [3, 0, 3, 0 . . . . . 2, 4, 2] → The Nos for 5 clusters, index start from 0 i.e [0-4]

5 Visualizing the clusters

↳ this is bcz 1st cluster is represented by 0

2D plot ka X co-ordi is 0th colm of X

2D plot ka y co-ord is 1th col of X

for cluster 1 i.e [0]

```
plt.scatter(X[y_kmeans==0, 0], X[y_kmeans==0, 1], s=100, c='red', label='Cluster1')
```

for 2

```
plt.scatter(X[y_kmeans==1, 0], X[y_kmeans==1, 1], s=100, c='blue', label='Cluster2')
```

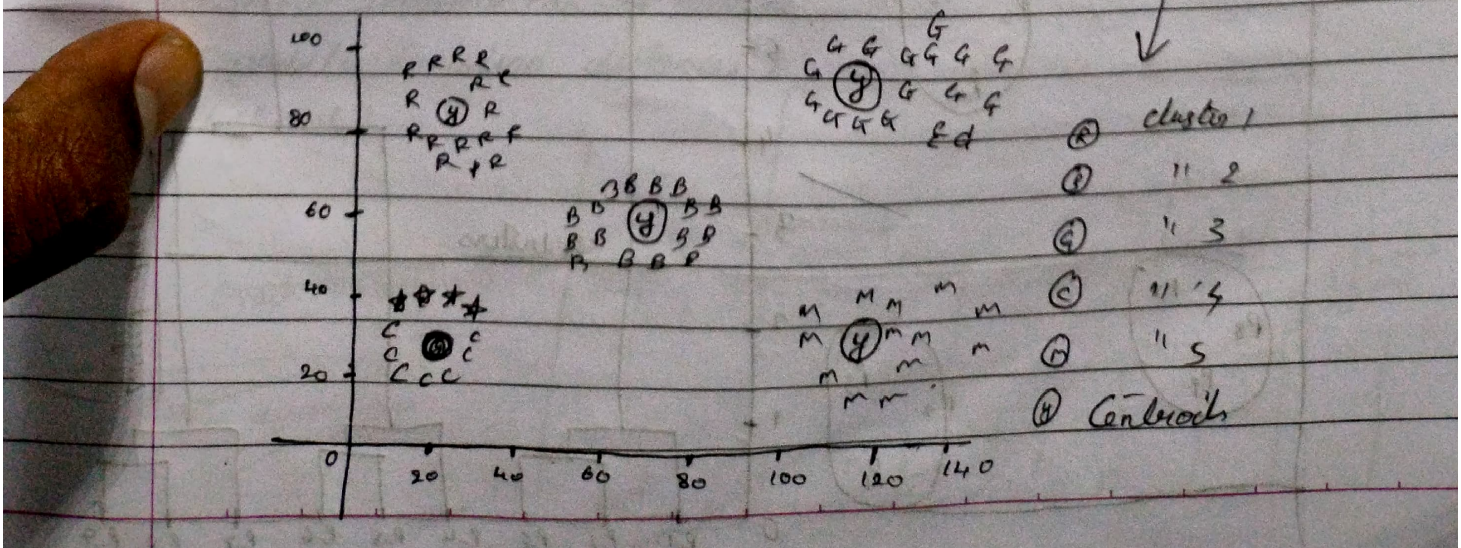
for 3 " " " 2, " " " " 2, " " c='green', "=Cluster3

for 4 " " " 3 3 c='cyan' 4

for 5 4 4 c='magenta' " 5

for Centroid

```
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1]
            , s=300, c='yellow', label='Centroids')
plt.title('Clusters of Customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



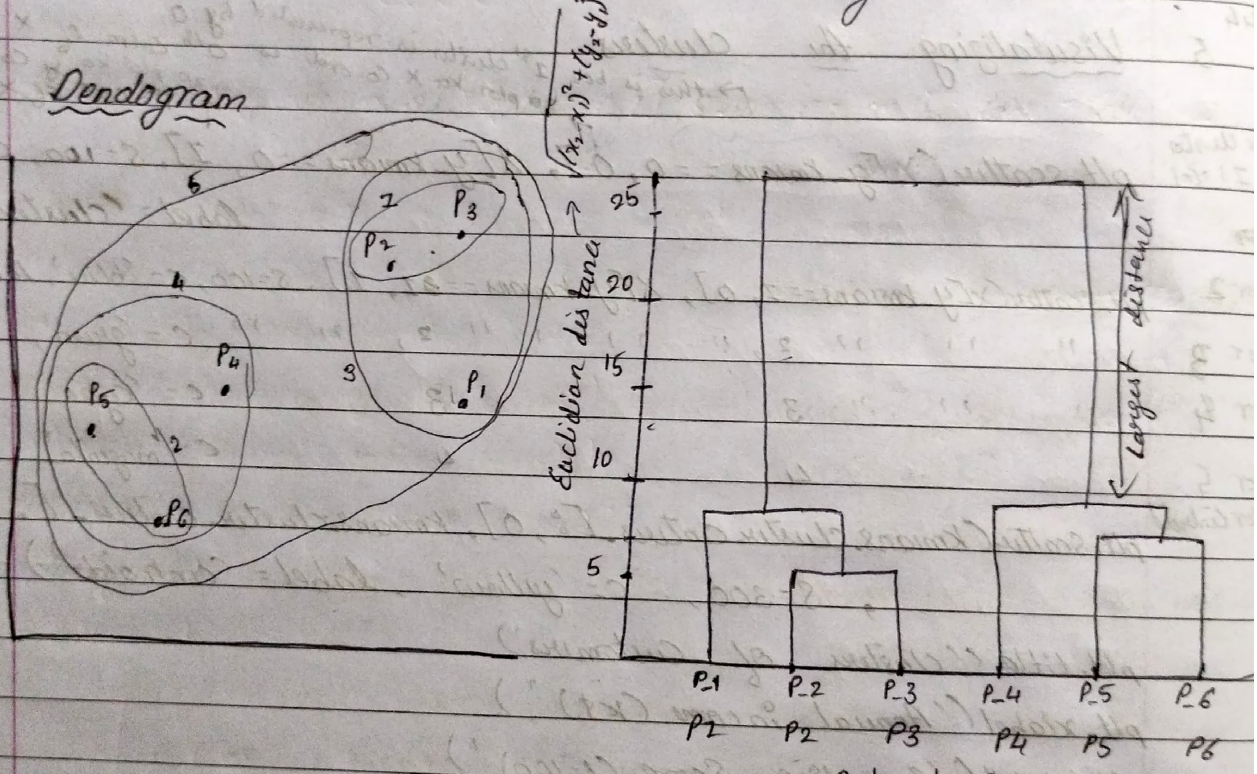
B] ☆ Hierarchical Clustering Intuition → Dendrome graph is used to find no. of clusters

Types
1] Agglomerative HC
2] Divisive

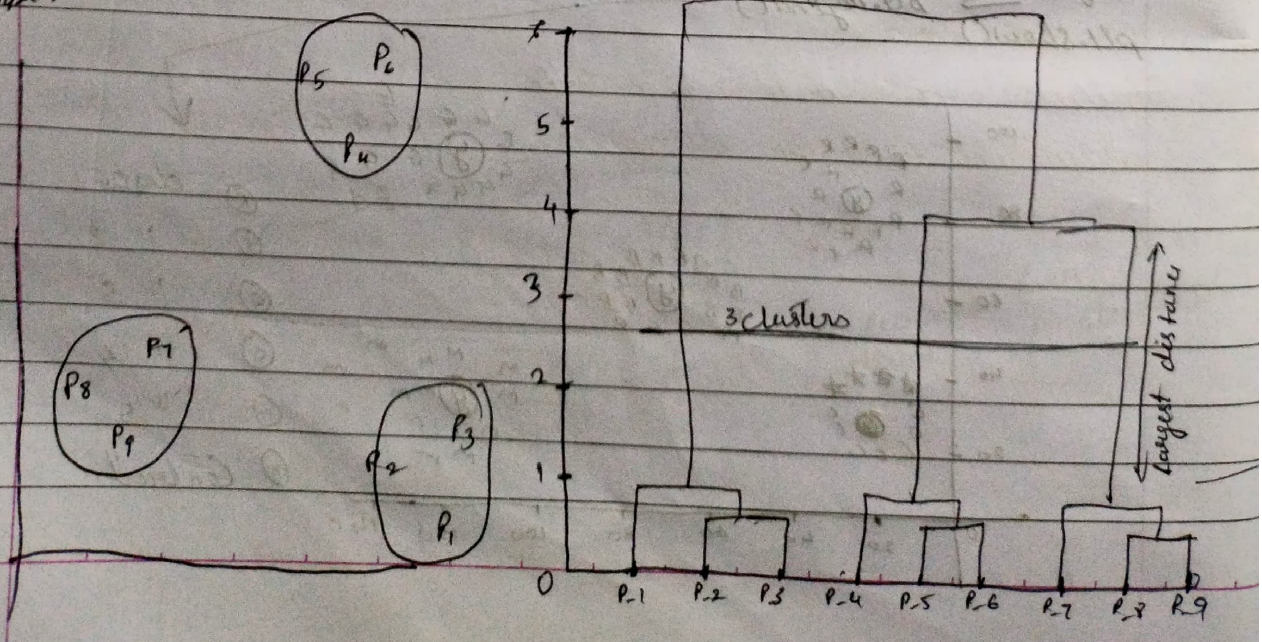
Agglomerative HC

Step 1 Make each data pt a single pt cluster → forms N clusters
2 Take 2 closest data pt & make them 1 cluster → forms N-1 clusters
3 " " " cluster & " " " " → forms N-2 "
☆ 4 Repeat Step3 until there are only 1 cluster.

Dendrogram



Example:



Page No.
Date

Steps

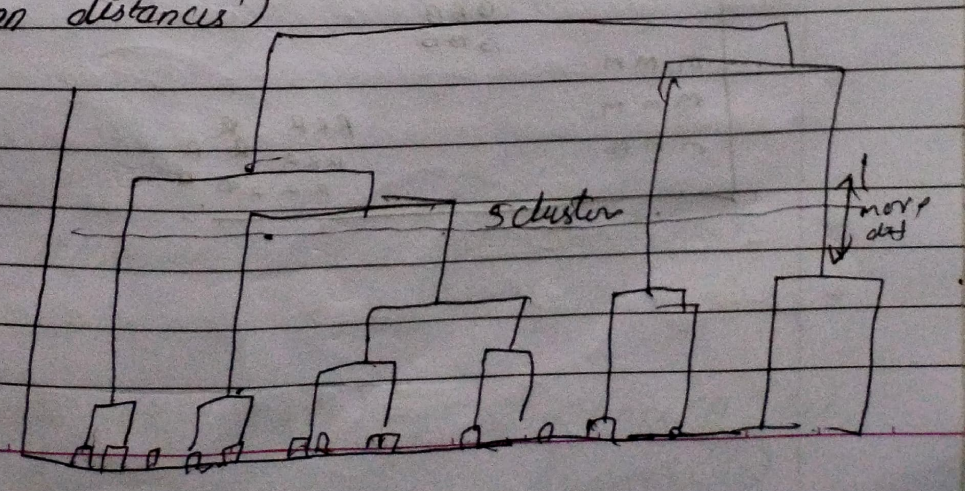
1 First all data pt are considered as cluster
2 Join pt which are closer & make them on cluster. Joining is marked as horizontal line on the graph. & Vertical line depicts the euclinoid ~~de~~ Eucladian distance btwn those 2 pts
3 Rpeat the steps & find a single cluster
4 Now Find Vertical line which is tall & dont cross any horizontal line
5 Once found that Vertical line draw a horizontal line See how many pts it cuts. Here its cutting 3pts hence total 3 clusters can be formed.

Steps for HC

1 Importing Libraries
2 " Dataset

3 Using the dendogram to find the optimal num of clusters

import scipy.cluster.hierarchy as sch
dendogram = sch.dendogram(sch.linkage(X, method = 'ward')) ← this minimizes the Variance
plt.title('Dendogram')
plt.xlabel('Customer no. of data pts')
plt.ylabel('Euclidian distances')
plt.show()



Graph is similar to this

This dendogram

If there is larger dataset then use K-Mean

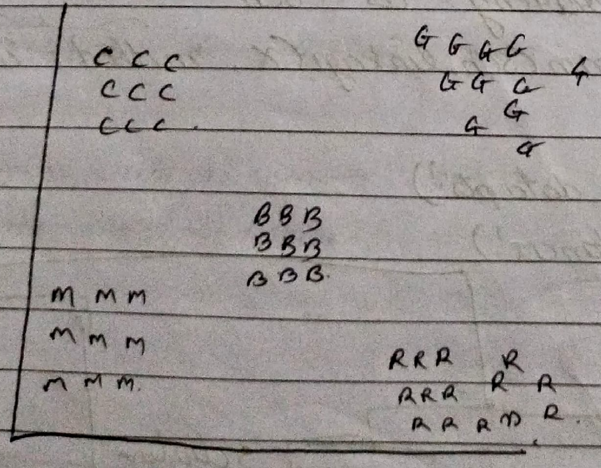
04, 5.6.

4 Training the Hierarchical clustering model on dataset

```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidian',
                             linkage = 'ward')
y_hc = hc.fit_predict(x)
```

5 Visualising the clusters

```
plt.scatter(X[y_hc==0, 0], X[y_hc==0, 1], s=100, c='red', label = 'Cluster 1')
  "       "          1              1          'blue'            2
                     2              2          'green'           3
                     3              3          'cyan'            4
                     4              4          'maginta'         5
plt.title('clusters of customers')
plt.xlabel('Annual income (k$)')
plt.ylabel('Spending score (1-100)')
plt.legend()
plt.show()
```



Part 6

~~Apr~~. Association Rule : People who bought this also bought that

A) Apriori Rule

B) Eclat.

A] Apriori Rule → How it contain
- Support
- Confidence
- lift

Apriori intuition

Apriori Support

- Movie Recommendation : $Support(M) = \frac{\#User\ watchlists\ Containing\ M}{\#User\ watchlists}$
- Market Basket Optimisation : $Support(I) = \frac{\#transaction\ Containing\ I}{\#transaction}$

Apriori Confidence

- Movie Recomendation : $Confidence(M_1 \rightarrow M_2) = \frac{\#User\ watchlist\ Containing\ M_1\ \&\ M_2}{\#User\ watchlist\ Containing\ M_1}$
- Market Basket optimisation : $Confidence(I_1 \rightarrow I_2) = \frac{\#transaction\ Containing\ I_1\ \&\ I_2}{\#transaction\ Containing\ I_1}$

Apriori - Lift

Movie Recommendation : $Lift(M_1 \rightarrow M_2) = \frac{Confidence(M_1 \rightarrow M_2)}{Support(M_2)}$

Market Basket Optimisation : $lift(I_1 \rightarrow I_2) = \frac{Confidence(I_1 \rightarrow I_2)}{Support(I_2)}$

Steps

1 Importing Libraries

```
!pip install apyori

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

2 Data processing

```
dataset = pd.read_csv('Market_Basket_Optimisation.csv', header=None)
transactions = []
for i in range(0, 7501):
    transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])
```

→ apyori dont take pd so we have to do list
→ This will access row
This is for each column of a row
→ the data should be in string.
Here the transactions is the list of list i.e the big list containing list of each row

→ [[, , , , ,], [. . . .], [. . .]]

3 Training the Apriori model on the dataset

```
from apyori import apriori
rules = apriori(transactions = transactions, min_support = 0.003,
                min_confidence = 0.2, min_lift = 3, min_length = 2,
                max_length = 2)
```

→ no. of time the product appe in transaction ÷ tot no o transactio i.e $\frac{3 \times 7}{7501}$ → 7 da
0.8 is ideal but try 0.8 to 0.1 & check which fit good work on that
Put this in btw 3 to 10 test then 3 is not good
often i.e buy 1 get 1 free buy 3 get 2 = 4, 3

4 Visualizing results

i] displaying the 1st results coming directly from o/p of apriori fun

```
results = list(rules)
results
```

ii] Putting the results well organised into Pandas DataFrams

```
def inspect(results):
    lhs = [tuple(result[2][0][0])[0] for result in results]
    rhs = [tuple(result[2][0][1])[0]  "   "   "   "  ]
    Supports = [result[1] for result in results]
    Confidences = [result[2][0][2]  "   "   "   "  ]
    Lifts = [  "  [2][0][3]   "   "   "  ]
    return list(zip(lhs, rhs, Supports, Confidences, Lifts))
resultsinDataFrame = pd.DataFrame(inspect(results), Columns =
        ['Left hand Side', 'Right Hand Side', 'Support', 'Confidence
                                    , 'Lifts'])
```

iii] Displaying the result non-Sorted

```
resultsinDataFrame
```

iv] Display result in Sorted way & in descending order

```
resultsinDataFrame.nlargest(n=10, Columns = 'Lift')
```

↳ Returns best 10 rows

↳ this fun tells which col^n to arrange in descend

3 Eclat : Here it only contain Support

* Movie Recommendation : $Support(M) = \frac{\#\text{User watchlists Containing } M}{\#\text{User watchlists}}$

* Market basket Optimisation : $Support(I) = \frac{\#\text{transactions Containing } I}{\#\text{transactions}}$

Page No.
Date

Steps

1 Importing Libraries
2 Data Preprocessing
3 Training Eclat model

4 Visualizing Result

i] Display 1st result - - - -

ii] Putting results ~~in~~ organised into pd dataframe

```
def inspect(results):
    lhs = [tuple(result[2][0][0])[0] for result in results]
    rhs = [tuple(result[2][0][1])[0]  "   "   "   "   "]
    Supports = [result[1] for result in results]
    return list(zip(lhs, rhs, Supports))
resultsinDataFrame = pd.DataFrame(inspect(results), Columns = ['Product 1',
                                  'Product 2', 'Support'])
```

iii] Display result in Sorted descending Support

```
resultsinDataFrame.nlargest(n=10, Columns = 'Support')
```

Part 7 Reinforcement Learning

1] Upper Confidence Bound Algorithm } Solve multi armed
2] Thomson Sampling } bandit problem
↳ more reliable

2] Upper Confidence Bound Algorithm

Step 1 At each round n, we consider two no. for each ad i

*) $N_i(n)$ - the no. of times the ad i was selected up to round n

*) $R_i(n)$ - the sum of rewards of ad i up to round n

Step 2 From these 2 num we compute

*) The avg reward of ad i up to round n

$$\bar{r}_i(n) = \frac{R_i(n)}{N_i(n)}$$

*) the confidence interval $[\bar{r}_i(n) - \Delta_i(n), \bar{r}_i(n) + \Delta_i(n)]$ at round n

with $$\Delta_i(n) = \sqrt{\frac{3}{2} \frac{\log(n)}{N_i(n)}}$$

Step 3 We select the ad i that has max: UCB $\bar{r}_i(n) + \Delta_i(n)$

Steps

1 Importing Libraries

2 Importing Dataset

```
dataset = pd.read_csv('Ads_CTR_optimisation.csv')
```

→

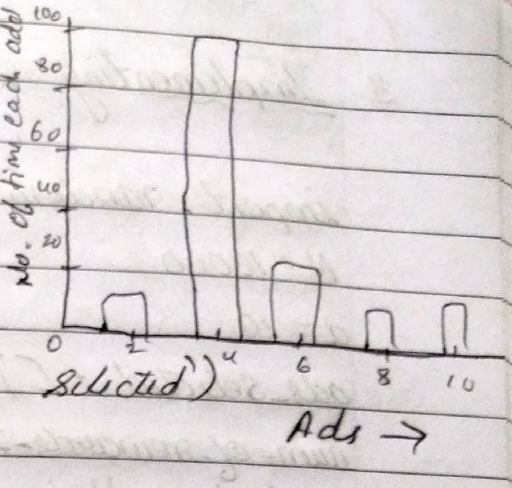
3 Implimenting UCB

```
import math
N = 10000
d = 10
ads_selected = []
numbers_of_selections = [0] * d
sums_of_rewards = [0] * d
total_reward = 0
for i in range(0, N):
    ad=0
    max_upper_bound = 0
    for i in range(0, d):
        if (numbers_of_selections[i] >0):
            average_reward = sums_of_rewards[i] / numbers_of_selections[i]
            delta_i = math.Sqrt()
            upper_bound = average_reward + delta_i
        else:
            upper_bound = 1e400
        if (upper_bound > max_upper_bound):
            max_upper_bound = upper_bound
            ad = i
    ads_selected.append(ad)
    numbers_of_selections[ad] = numbers_of_selections[ad] + 1
    reward = dataset.values[n, ad]
    sums_of_rewards[ad] = sums_of_rewards[ad] + average_reward
    total_reward = total_reward + reward
```

Page No.
Date

4 Visualizing the Result

```
plt.hist(ads_selected)
plt.title('Histogram of ads selections')
plt.xlabel('Ads')
plt.ylabel('Num of times each ad was selected')
plt.show()
```



2] Thomson Sampling : It is Probabilistic algo

Step 1 At each round n, we consider 2 num for each ad i
- $N_i^1(n)$ - the num of time ad i got reward 1 up to round n
- $N_i^0(n)$ - " " " " " " " " 0 " " " "

Step 2 For each ad i, we take a random draw from the dist^n below

$$\theta_i(n) = \beta(N_i^1(n)+1, N_i^0(n)+1)$$

Step 3 We select ad that has highest $\theta_i(n)$

Steps

1 Importing libraries

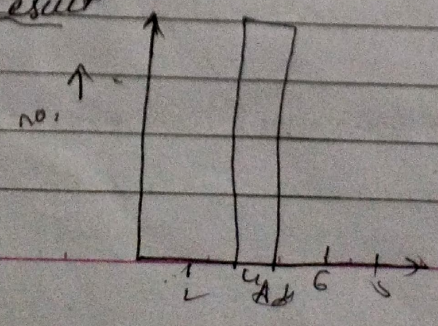
2 Importing dataset

→

3 Implementing Thompson Sampling

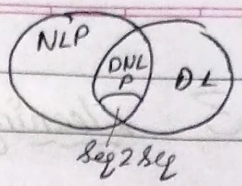
```
import random
N = 10000
d = 10
ads_selected = []
num_of_rewards_1 = [0] * d
num_of_rewards_0 = [0] * d
total_reward = 0
for n in range(0, N):
    ad = 0
    max_random = 0
    for i in range(0, d):
        random_beta = random.betavariate(num_of_rewards_1[i] + 1, num_of_rewards_0[i] + 1)
        if (random_beta > max_random):
            max_random = random_beta
            ad = i
    ads_selected.append(ad)
    reward = dataset.values[n, ad]
    if reward == 1:
        num_of_rewards_1[ad] += 1
    else:
        num_of_rewards_0[ad] += 1
    total_reward = total_reward + reward
```

4 Visualizing the Result



Page No.
Date

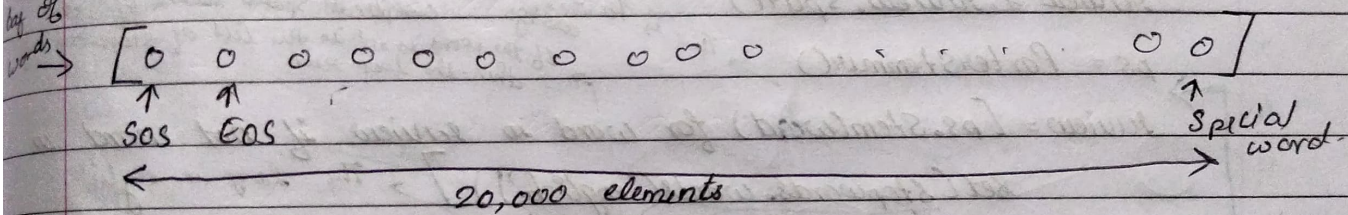
Part 7 - Natural Language Processing



This applies ML model to text & language.

Bag of words

Here in general there are 171476 words So our text is converted into numbers (i.e no. of time our word gets repeated). Likewise similarly training set text are also converted into num & model is trained The limitation is that we can detect only yes or no & not much complex response.



Natural Language processing

Steps

1 Importing the libraries

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

2 Importing dataset

```
dataset = pd.read_csv('Restaurant_Reviews.tsv', delimiter='\t'
                      , quoting=3)
```

↳ to avoid Sentiment analyses i.e " "

1→2 −1→0→21
0→1 0→1→100
<1→0 1→2→100

[chay . is . good]

K = K → 0 → 2
0 = 2 → 1 → 100
1 = 0 → 2 → 80

Page No.
Date

306

3 Cleaning the texts

```
import re
import nltk                                   → for stopwords like the, a, ....
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer    → This is for loved or love Er
                                                I loved this = I love this
corpus = []
for i in range(0, 1000):                      → replace punctuation by space
    review = re.sub('[^a-zA-Z]', ' ', dataset['Review'][i])
                  → This line uses regular expression to replace all the punctuation
    review = review.lower()
    review = review.split()                   → to create token i.e loved = love, so that we dont create complicated sparse matrix of words
307
    ps = PorterStemmer()                      → This is
    review = [ps.stem(word) for word in review if not word in
              set(stopwords.words('english'))]   → The lang is english
                  → If the word is not in the list of stopword only then the loop run
    review = ' '.join(review)
    corpus.append(review)
```

replace this by all_words

replace

```
all_words = stopwords.words('english')
all_words.remove('not')
```

→ This bcz it was considering not as stop word & removing it

309

4 Creating the Bag of words model

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features = 1500)
X = cv.fit_transform(corpus).toarray()
y = dataset.iloc[:, -1].values
```

→ here max length of our bag of words will be 1500 i.e it will again remove some of the non stop words like num chars.

→ This will convert those bow into columns

→ To check the o/p for single review check the Bonus lecture

5 Splitting into training & Test

6 Training Naive Bayes on training

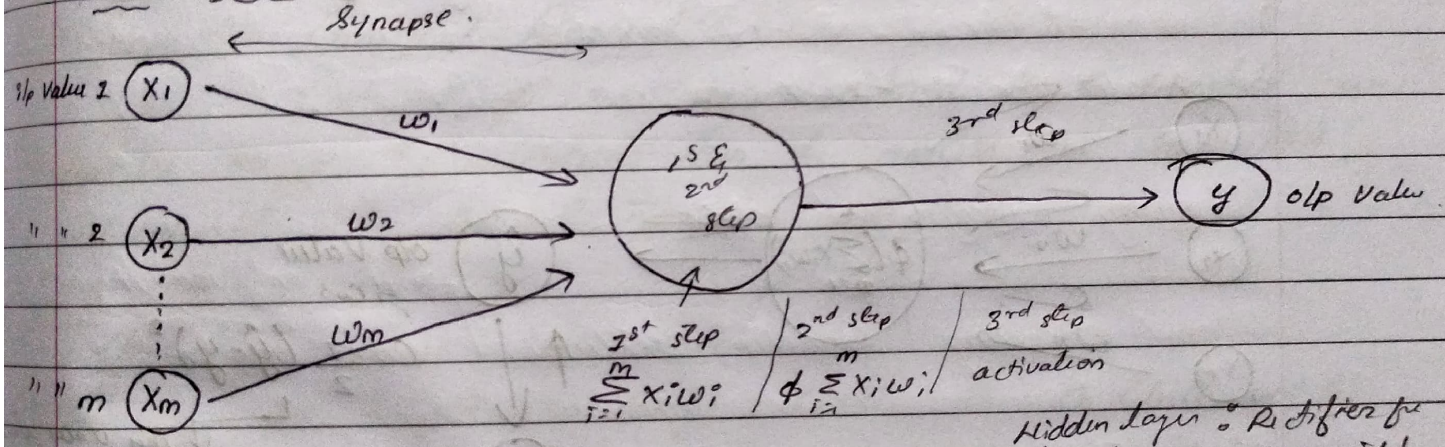
7 Predicting Test

8 Making Confusion

Part 8 **Deep Learning** (Geoffrey Hinton)

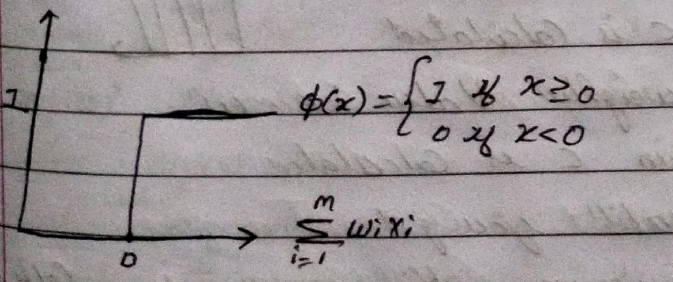
1 **Neural Network**

The Neuron

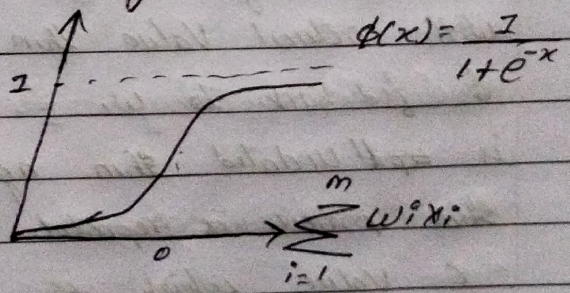


Activation fun^n

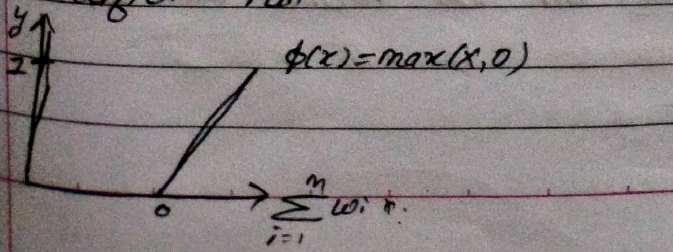
i] Threshold Fun^n



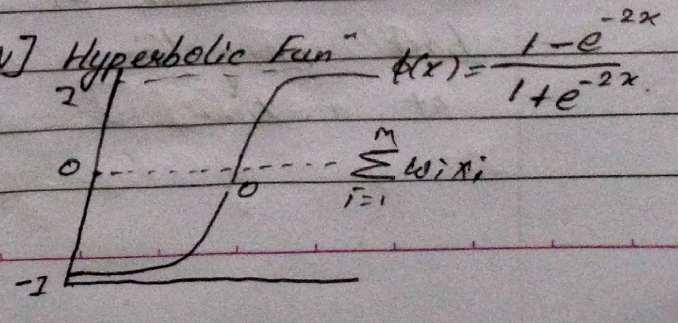
ii] Sigmoid Fun^n



iii] Rectifier Fun^n



iv] Hyperbolic Fun^n



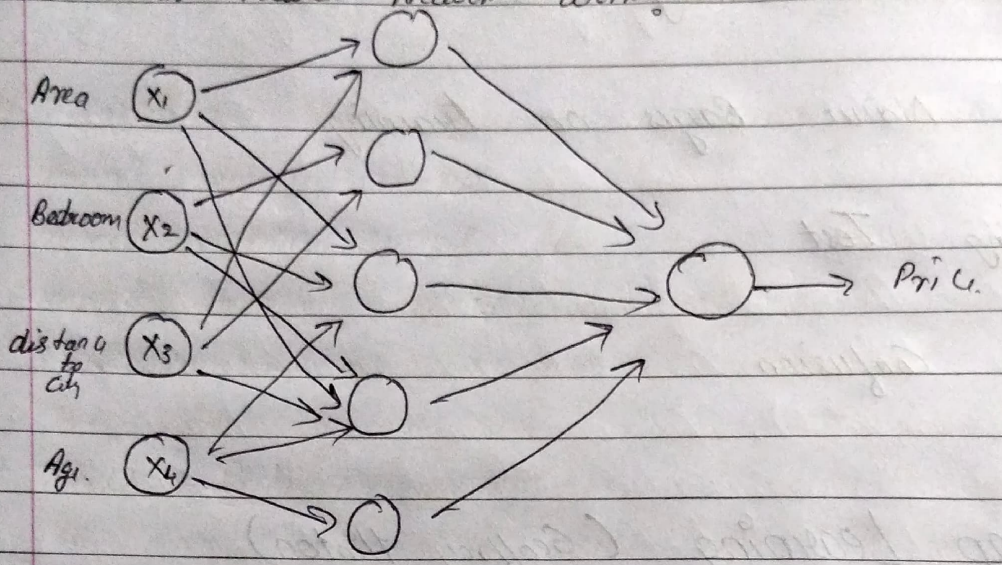
2.32
12.47

Page No.
Date

best intuition

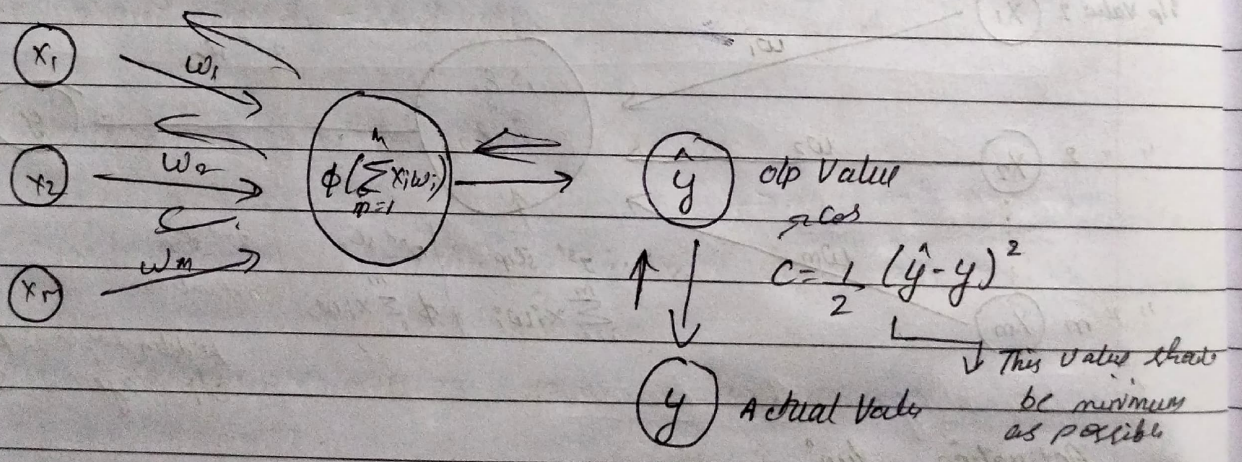
329

How do Neural Network work?

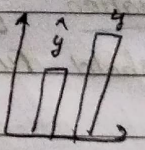


330

How do Neural Networks learn?



Here initially the olp value is compared with actual value then C is calculated & fed back to w_1 ∴ weighted Value i.e w_1 is updated then again C is calculated this process is repeated until you get C Value which is minimum this process is called back propagation. Here all the rows are updated at a time

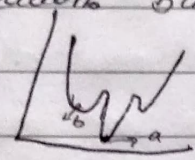


331

332

331 Gradient Descent

Here we just chck which way is downhill & thruby going in that direction
In this method all the rows of dataset are chckd or monitored at a time & updated thruby repeating process of back propogation. But wt if we had a curve like this →



Here 'a' will be global minimum but our ball will be stopped at 'b' i.e local minimum
∴ Here comes Stochastic Gradient

332 Stochastic Gradient Descent

Here Single row is monitored & E weights are updates as per 'c' value. Then the process is repeated individually to all the rows.

Steps in training ANN model

1 Initialize the weight randomly close to 0 (but not 0)
2 Ip the 1st observation of dataset, each feature in 1 i/p node
3 Forward propogation thruby finding $\hat{y}$
4 Compare $\hat{y}$ with y & find the error i.e 'c'
5 Back propogation, update the weights as per error
6 repeat Stp 1 to 5 after each observation until you get less error i.e Conv
7 When whole training Set pass through ANN that makes an epoch. Redo more epochs

Steps

1 Libraries

```
import numpy as np
import pandas as pd
import tensorflow as tf
tf.__version__
```

2 Part 1 : Data Preprocessing

i] Importing dataset

```
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
```

ii] Encoding Catagorical data

// Label encoding "Gender" Column → Here only male/Female was there hence we used Label encoder

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

→ In the main dataset the Gender column is at index '5'. But we have dropped or not considered some colm while importing dataset. So Change this index value as per our modified X Value & not the actual dataset

// One Hot Encoding for "Geography" Column

```
from sklearn.compose import ColumnTransformer
from ".preprocessing" import OneHotEncoder
ct = ColumnTransformer(transformers = [('encoder', OneHotEncoder(),
                [1])], remainder = 'passthrough')
X = np.array(ct.fit_transform(X))
```

Page No.
Date

iii] Splitting the dataset
//same

iv] Feature Scalling
//same
} It is must in deep learning. We have to apply feature Scalling to all the columns

3 Part 2 - Building the ANN

i] initializing the ANN
ann = tf.keras.models.Sequential()
This formed the seq i.e ip, hidden layer & op

ii] Adding the ip & first hidden layer
6 neuron
these are hyperparameter.
It's code word for rectifier.
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
↳ It is rectifier i.e relu for ip & Sigmoid for op

iii] Adding the 2nd hidden layer
num of neuron
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))

iv] Adding the op layer
when we have binary op use 1 if we have more than 2 op then used the length of op ex if 3 op i.e 0, 1, 2 then units=3
ann.add(tf.keras.layers.Dense(units=1, activation='Sigmoid'))
↳ If we had non-binary value then activation is Softmax

Part 3 - Training the ANN

i] Compiling the ANN
updates weight
for non-binary dp this should b/ chatorical-CrossentropY
ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['Accuracy'])

ii] Training ANN on training set
ann.fit(X_train, y_train, batch_size=32, epochs=100)

5 Part 4 - Making the predictions & evaluating the Model

i] Predicting the result of a single observation

print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) > 0.5)

(France)

Here we used sc.transform bcz earlier while training our model we have scaled our model so we should use scalled value to predict the new observation.

We have onehotencoded France func we have to insert dummy variable first. bcz dummy var takes 1st place in the col. Furthermore we used 0.5 as threshold if value > 0.5 then 1 (True) or 0 (False). Because predict method always takes 2D : [[...]]

Part 9

Section 38 Dimensionality Reduction Technique

To reduce the num of independent features we use Dimensionality Technique

Types

1 Feature Selection
2 Feature extraction

Section 39 Principal Component Analysis (PCA) (unsupervised)

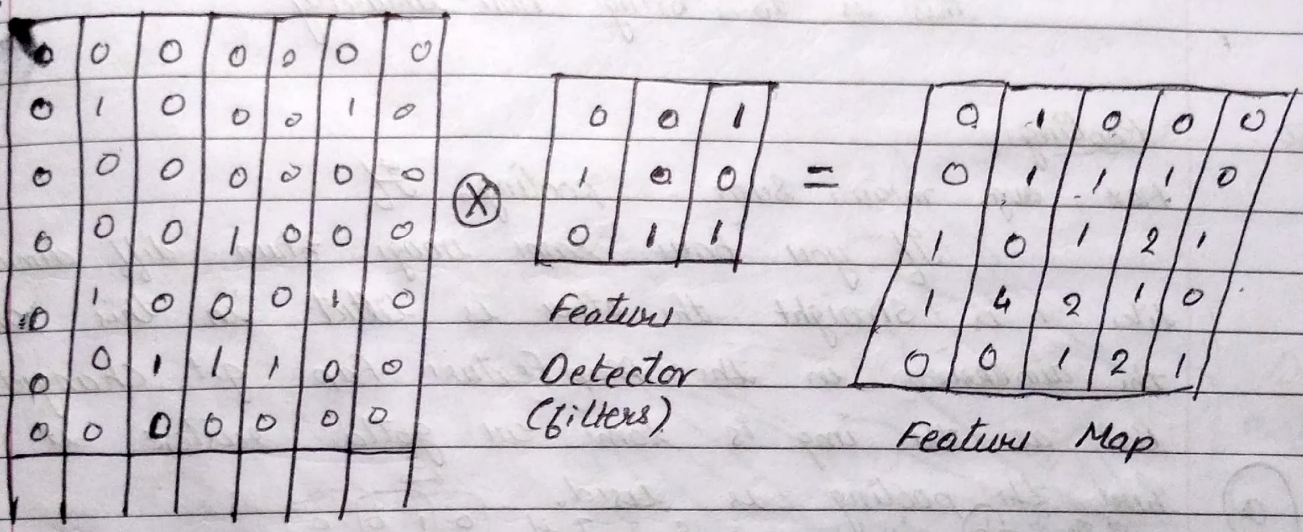
↳ Used to reduce the size of independent var

It reduce the dimensions of a d-dimensional dataset by projecting it onto a (k) - dimensional subspace (where $k < d$)

Page No.
Date

Convolution Neural Network (Yann Lecun)

348



Input Image

Feature Detector (filters)

Feature Map

1 img = 0-255 pixels

Feature Mapping helps in reducing the size

CNN is to extract the features of an img & gives feature map

i] Sharpen: EO

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

ii] Blur

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

iii] Edge Enhance

0	0	0
-1	1	0
0	0	0

iv] Edge detector

0	1	0
1	-4	1
0	1	0

v] Emboss

-2	-1	0
-1	1	1
0	1	2

349 ReLU [Rectifier Linear Unit]

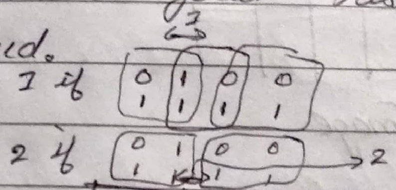
This is to bring non-linearity.

350 Pooling

Max, avg, mean, sum :. pooling. If

If you have same image then diff dimentions like, 1 is straight, the other is tilted in this case the dimension in the ~~Map~~ Feature Map get changed. But as the img is same we gotta restore it here the pooling is used. (a)

1st stride (b) 2nd stride → Ex: 1st



2×2 is used for pooling

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

Max pooling →

1	1	0
4	2	1
0	2	1

Pooled Feature Map

351 ~~Fa~~ Flattening

1	1	0
4	2	1
0	2	1

Pooled Feature Map

→

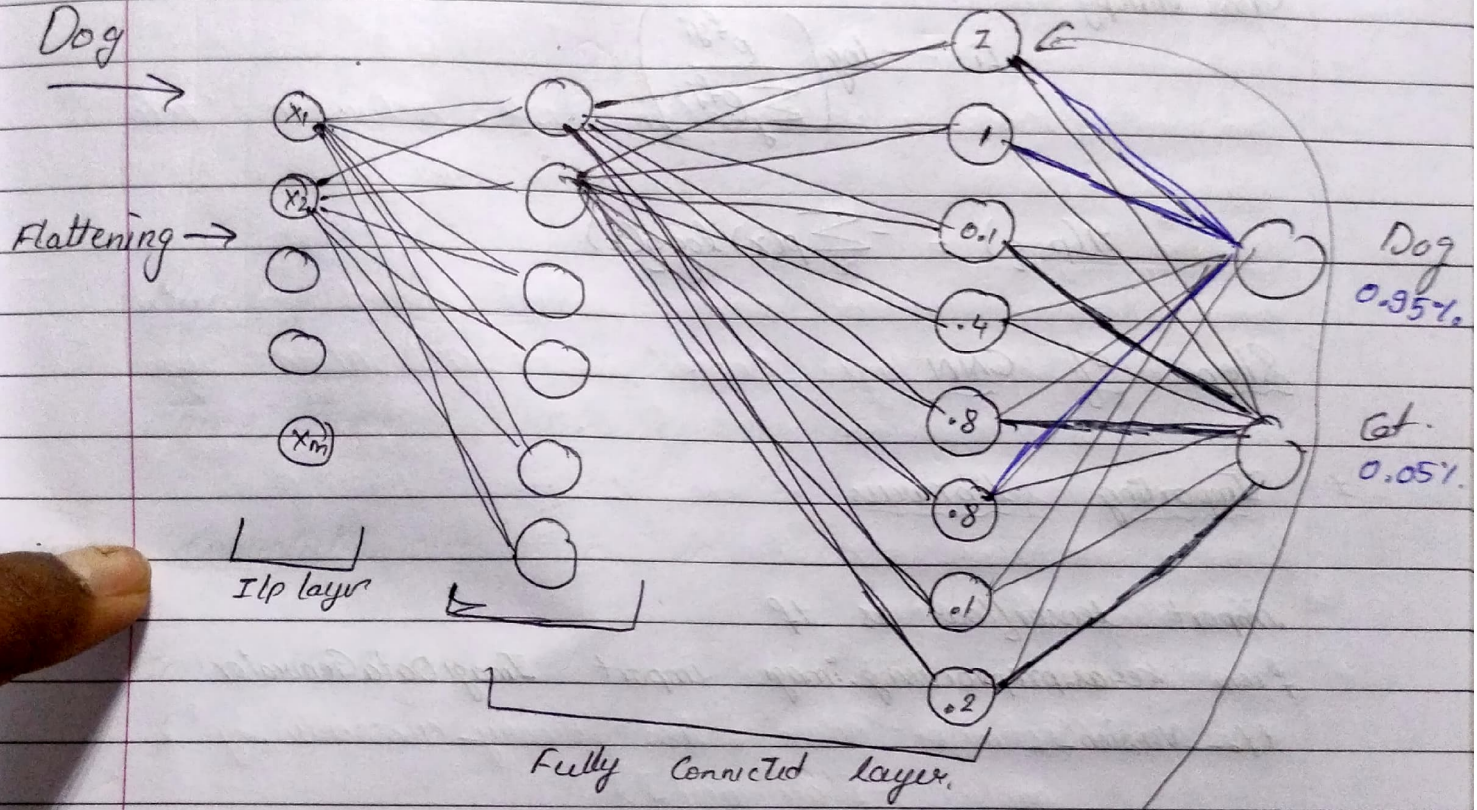
1
1
0
4
2
1
0
2
1

Note:- In ANN the hidden layer is not restricted to be fully connected

In CNN the hidden layer must be fully connected.

Input Image → Convolution → ReLU → Pooling → Flattening

352 Full Connection



Here Dog is given ip every neuron will hold certain feature like eye, ear, nose etc. The 1st neuron contains 1 value is sent to both o/p. Now it's on both o/p to take a decision. The lets say first neuron has info abt eye, the eye is sent to both o/p but now o/p dog will identify it & mark a pt that neuron 1 belongs to him. Cat identifies 1st doesn't belong to him. If any error happened then it's back propogated & feature detector or filters value is adjusted accordingly.

353 Just for a quick glance of CNN you can refer this to Udo

60
Train 8K
Test 2K

Page No.
Date

354 Softmax fun^n

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Cross entropy

$$Li = -\log\left(\frac{e^{fy_i}}{\sum_j e^{f_j}}\right)$$

$$H(p,q) = -\sum_x p(x) \log q(x)$$

Steps of CNN

1 Importing Libraries

Step 1

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
tf.__version__
```

2 Data Pre-processing

i] Pre-processing the training set

Step 2

```
train_datagen = ImageDataGenerator(
        rescale = 1./255,          → pixel are btwn 0-255 so that they can scale btwn 0 & 1.
        shear_range = 0.2,
        zoom_range = 0.2,
        horizontal_flip = True)    → This is to connect the above instance with our dataset
training_set = train_datagen.flow_from_directory(
        'dataset/training_set',    → Path of the file
        target_size = (64, 64),
        batch_size = 32,           → It means 32 img will be there in 1 batch
        class_mode = 'binary')     → bcz we have only 2 o/p i.e dog & cat
```

to avoid overfitting img augmentation is done i.e size of img is pre-determined

Page No.
Date

ii] Preprocessing Test Set

↗ for test-set should apply only this i.e only feature scaling

```
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory(
        'dataset/test_set',
        target_size = (64, 64),
        batch_size = 32,
        class_mode = 'binary')
```

Building the CNN

```
cnn = tf.keras.models.Sequential()
```

Step1 Convolution

try diff values as well

feature detector matrix i.e 3×3 if 4×4 then 4

feature size : 3

```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size = 3,
        activation = 'relu', input_shape=[64,64,3]))
```

Rectifier

64 bcz earlier we have selected the target size as 64

we have Colored img : 3 if black & white img then add 1 instead of 3.

Step2 Pooling

how u should add the size of matrix u used during pooling for max; mentioned in diag

size at which u stride. mention as ⓑ in aigram

```
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Adding 2nd Conv layer

Add 2nd layer here u no need to add input_shape parameter

```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn
        activation = 'relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Step 3 Flattening

```
cnn.add(tf.keras.layers.Flatten())
```

Step 4 Full Connection

```
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

units = no. of neuron. as u want

Step 5 Output layer

```
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

for o/p add no. of neuron = size of o/p (in our case o/p is dog or cat so 1)

Part 3 Training the CNN

* Compiling the CNN

```
cnn.compile(optimizer='adam', loss='binary_crossentropy',
            metrics=['accuracy'])
```

* Training the CNN on the training set & evaluating it on the test set

```
cnn.fit(x=training_set, validation_data=test_set, epochs=25)
```

Part 4 Making a single prediction

```
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg',
                            target_size=(64, 64))
test_image = image.img_to_array(test_image)
```

u have to give the exact size of image which u have used in data preprocess.

↳ Convert that img to array

Page No.
Date

Earlier batch size was given 32, so now again we have to edit the dimension, axis=0 bcz the image is set in 0th batch.

```
test_image = np.expand_dims(test_image, axis=0)
result = cnn.predict(test_image)
training_set.class_indices → It will give us the binary value of cat & dog i.e 1 for dog & 0 for cat.
if result[0][0] == 1:
    print('dog')  → 0th bath & 0th element.
else:
    print('cat')
```

1 Principleal Componint analysis

Steps

1 Importing Libraries
2 Importing Dataset → wine.dataset
3 Splitting
4 Feature Scaling
5
5 Applying PCA

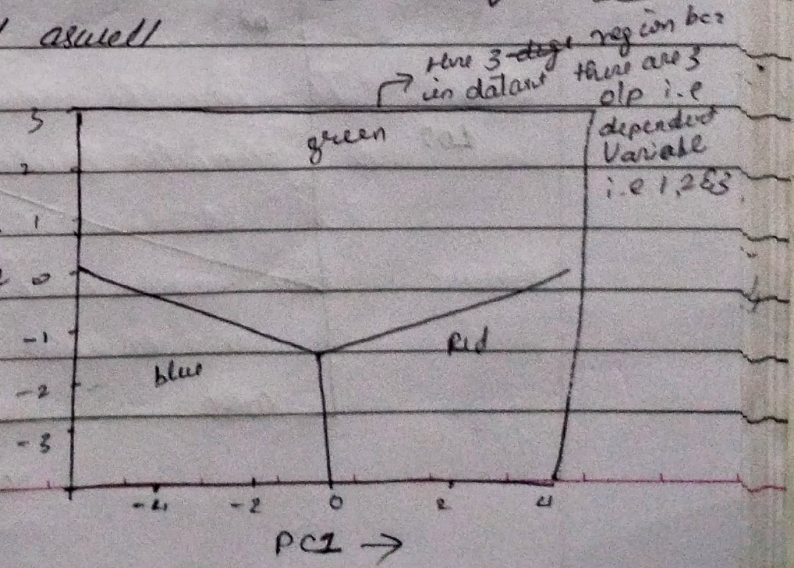
```
from Sklearn.decomposition import PCA
pca = PCA(n_components = 2) → you can set any of the num as much feature u want.
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

6 Training the logistic Regression model on the training set
// try with other model aswell

7 Making Confusion matrix
⇒ Take the code for this

8 Visualizing training set

9 " Test Set



accuracy = 97%.

→ Supervised

2 Linear Discriminant Analysis (LDA)

LDA & PCA are linear transformation technique

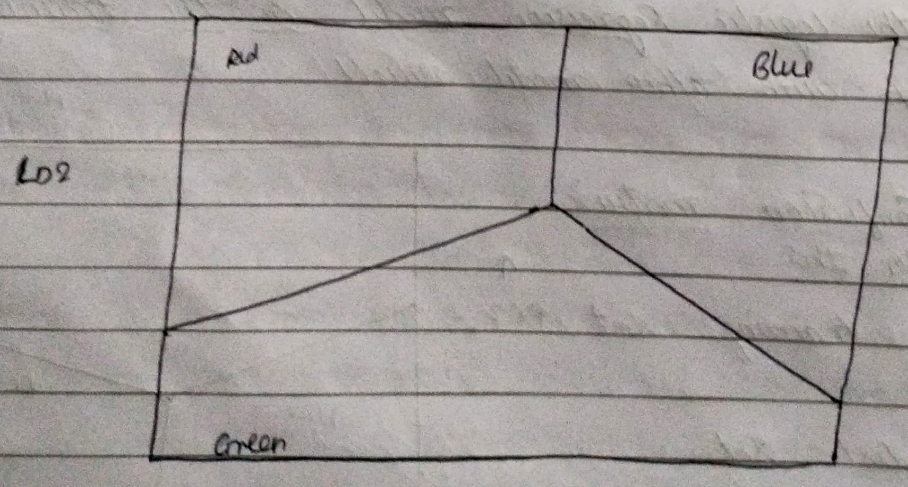
1 Importing Libraries
2 " dataset
3 Splitting
4 Scalling

5 Applying LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
                                                    as LDA
lda = LDA(n_components = 2) → 2 extracted feature
X_train = lda.fit_transform(X_train, Y_train) → used both X & Y train comes
X_test = lda.transform(X_test)
```

6 training with logistic model on training set
Itry with other method as well

7 Making Confusion matrix → got 100% accuracy
8 Visualizing training set
9 " Test set



Page No.
Date / /

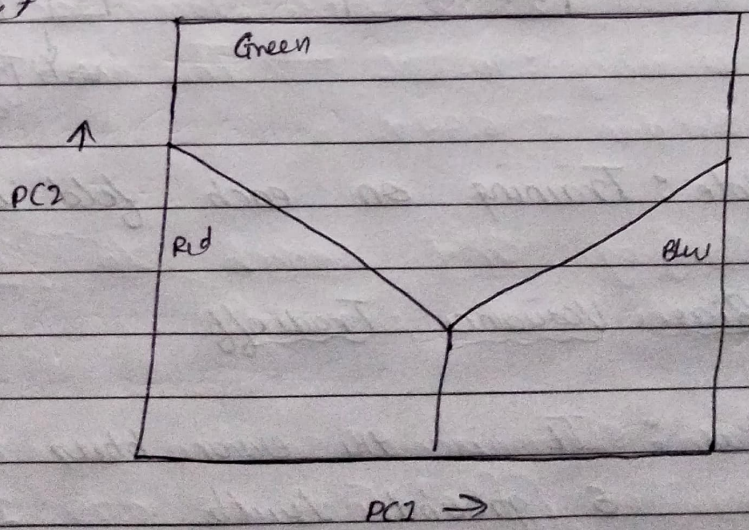
3 Kernel PCA

1. Importing Libraries
2. " Dataset
3. Splitting
4. Feature Scalling

5. Applying Kernel PCA

```
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(n_components = 2, kernel = 'rbf')   → Radial Basis func
X_train = kpca.fit_transform(X_train)
X_test = kpca.transform(X_test)
```

6. train with logistic on train set
7. Making Confusion matrix (100% accuracy)
8. Visualizing training set
9. " test set



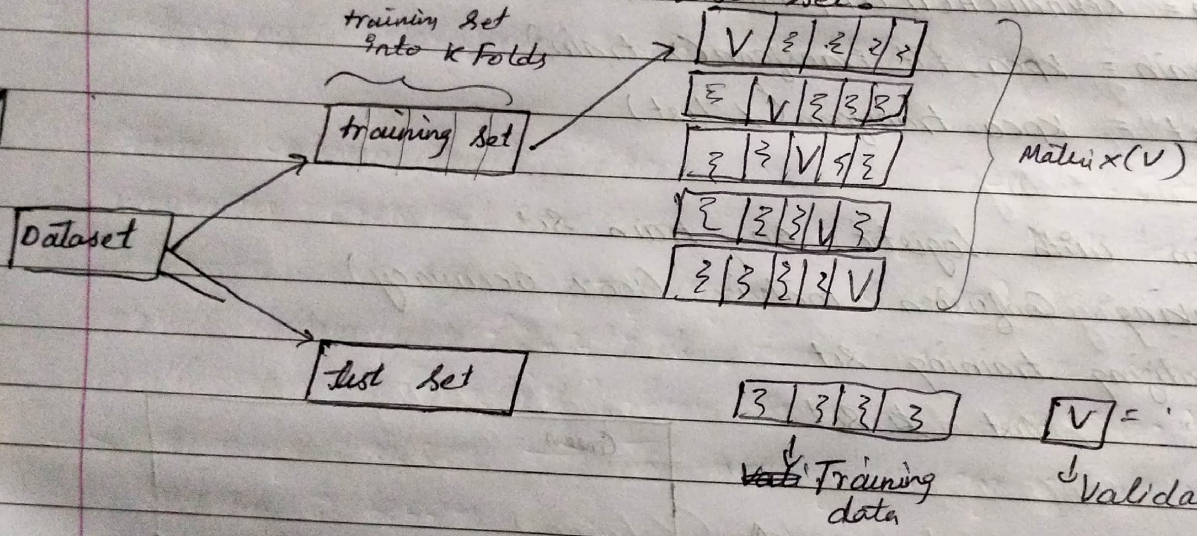
Overfitting : High Accuracy in training set & poor in test set

Page No.
Date

Part 10 Model Selection & Boosting

i] K- Fold Cross Validation

In classic method we just split data into training & test, then train the data onto training followed by test set. But what if you got lucky on that test set & the model doesn't hold good for another new data. For this case k-fold is used. Here the training data is divided into different matrix & model is trained on that model. If everything holds good then the model is retrained on checked on test set.

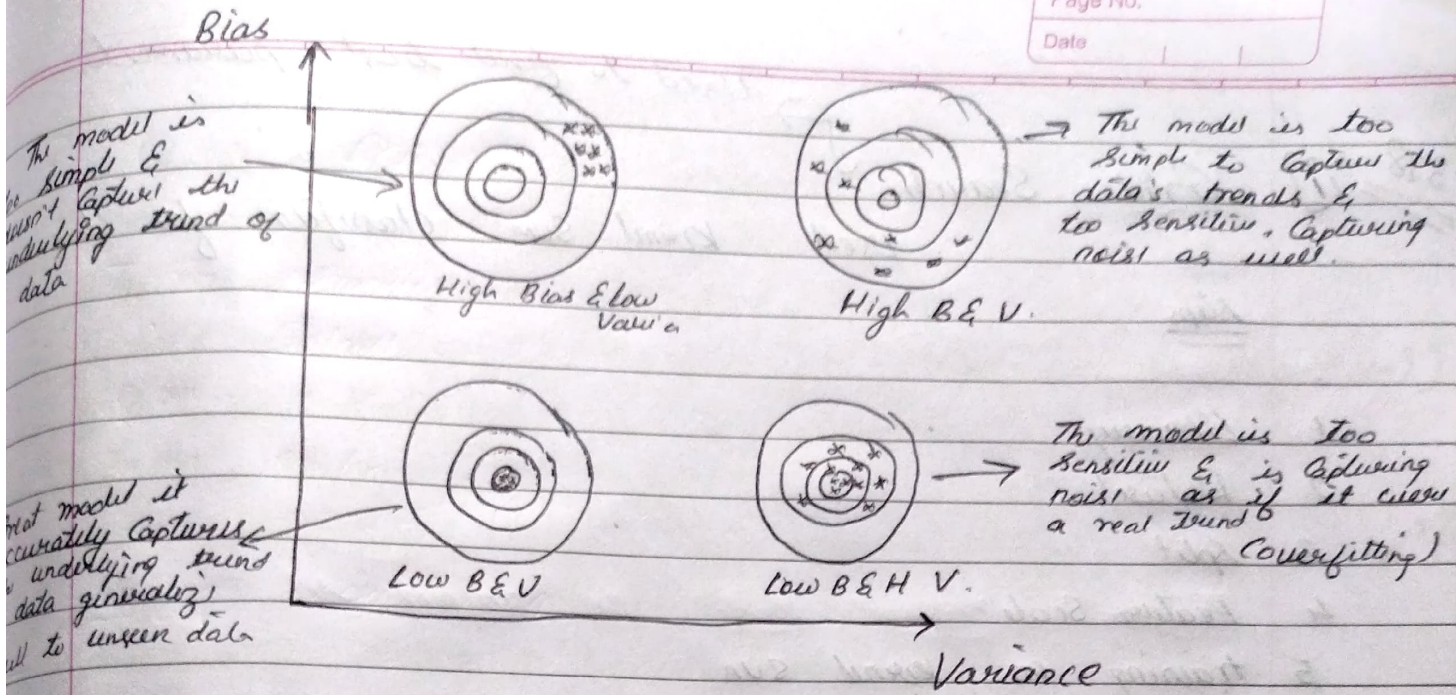


Note : Training on each fold uses the same hyperparameter

376 Bias Variance Tradeoff

Bias : It is the error btwn avg model prediction & ground truth.

Variance : How much the model can adjust depending on given dataset.



77 Steps

This is an kernel svm classifier model.

1 Importing Library
2 Dataset
3 Splitting
4 Feature Scaling
5 training model
6 Pred test set
7 Confusion matrix
8
8 Applying K-Fold Cross Validation

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
accuracies.mean()
accuracies.std()
```

→ This type this if u are in Spyder or anaconda ide if u are in g collab type this

cv → no. of folds

cv is applied to trainig set

Here we have to use the name of var we used during training more specifically var we fitted using X_train & y_train

9 Visualizing the training set result

```
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Std dev: {:.2f} %".format(accuracies.std()*100))
```

Page No.
Date

Used to find best parameter

378 ii] Grid Search

Used Kernel SVM classifier for this

Steps

1 Library
2 Dataset
3 Split
4 Feature Scale
5 training on Kernel SVM
6 Confusion matrix

7 Applying k-fold Cross Validation

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train
                             , y = y_train , cv = 10)
print("Accuracy : {:.2f} %".format(accuracies.mean()*100))
print("Std dev : {:.2f} %".format(accuracies.std()*100))
```

→ Accuracy : 90.33% → This accuracy is avg of all 10 folds
Std dev : 6.57%

8 Applying Grid Search to find the best model & the best parameter

```
from sklearn.model_selection import GridSearchCV
parameters = [{'C':[0.25, 0.5, 0.75, 1], 'kernel':['linear']},
              {'C':[0.25, 0.5, 0.75, 1], 'kernel':['rbf'],
               'gamma':[0.1, 0.2 ---- 0.9]}]
grid_search = GridSearchCV(estimator = classifier,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10,
                           n_jobs = -1)
```

→ should use Var which has been used for train specification which fit method is applied.

```
grid_search.fit(X_train, y_train)
best_accuracy = grid_search.best_score_
best_paramiters= grid_search.best_params_
print("Best Accuracy : {:.2f} % ".format(best_accuracy*100))
print("Best Paramiters: ", best_paramiters)
```

→ Best Accuracy : 90.67%.
Best paramiters : {'C':0.5, 'gamma':0.6, 'kernel':'rbf'}

Now while training this model we used SVC, so go to Scikit learn API & chick hyperparamiters or SVC, there are alot of them. 1st is 'C' paramiter i.e regularization paramiter which is in range of 0 & 1, ~~it is strong if C= its string~~ strength is inversely prop to 'C' strength ↑ at 0.25 strength ↓ at 1. , next paramiter is kernel you can choose linear, poly, rbf, Sigmoid or precomputed. 'Gamma', you can use this only if kernel is rbf, poly & Sigmoid, ranges btwn 0-1

* Now Here in paramiters we made a list of 2 dict one for linear kernel & other for rbf.
* grid_search contain classifier as estimator
* Scoring is set to accuracy as we are working on classification method
* param_grid is set to list of paramiter we chose earlier
* cv = 10 i.e 10 to 10 k-fold
* n_jobs = if it is set to -1 then it will use all the processor of CPU & consume less time to perform the complex operation.

→ Here you can see the best accuracy & hyperparamiters for the same

Page No.
Date

329 XGBoost

steps

1 Library
2 Dataset
3 Split
4
4 Training XGBoost Model on training set

```
from xgboost import XGBClassifier
classifier = XGBclassifier()
classifier.fit(X_train, y_train)
```

5 Confusion matrix → 97%.

6 Applying k-fold.
Accuracy = 96.53%
std dev = 2.07%.