60004210210

Amartya Mishra

COMPS – C31

ML Experiment 1

60004210210
Amartya Mishra
COMPS - C81

ML - Experiment - 1

Aim: To perform Data cleaning & preprocessing on the Dataset & implement Linear Regression

Theory:

Data cleaning is the process of fixing or removing incomplete, incorrect, corrupted, incorrectly formatted, duplicate or incomplete data within a dataset.

While Integrating Multiple data sources, discrepency is possible which makes Outcomes & algorithms unreliable.

Steps to clean Data:

- → Remove irrelevent data
- → Remove duplicate & incomplete cases
- → Fix Structural Errors
- → Deal with Missing Data
- → Identify & review outliers
- → Filter out outliers
- → Encode categorical data
- → Splitting the Dataset.
- → feature Scaling.

Linear Regression:

It is a special type of machine learning algorithm more specifically a supervised Algorithm. It learns from the labbled dataset & maps the data points to the most optimized linear functions

It computes the linear relationship between a dependent variable & one or more independent features.

Simple Linear Regression :

$y_i = \beta_0 + \beta_1 x$

Y: dependent variable
X: Independent variable
β_0: Intercept
β_1: Slope

The goal of Algorithm is to find the best fit line Equation that can predict the value based on independent variables

$$\text{cost function } (J) = \frac{1}{n} \sum_{i}^{n} (\hat{y}_i - y_i)^2$$

n → No. of datapoints
y_i → actual value
$\hat{y}_i$ → Predicted value

Conclusion : Thus we understood steps of Data pre-processing & cleaning & there after implemented linear regression.

Implementation: Done in 3 ways, firstly on given data using direct formulas.

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

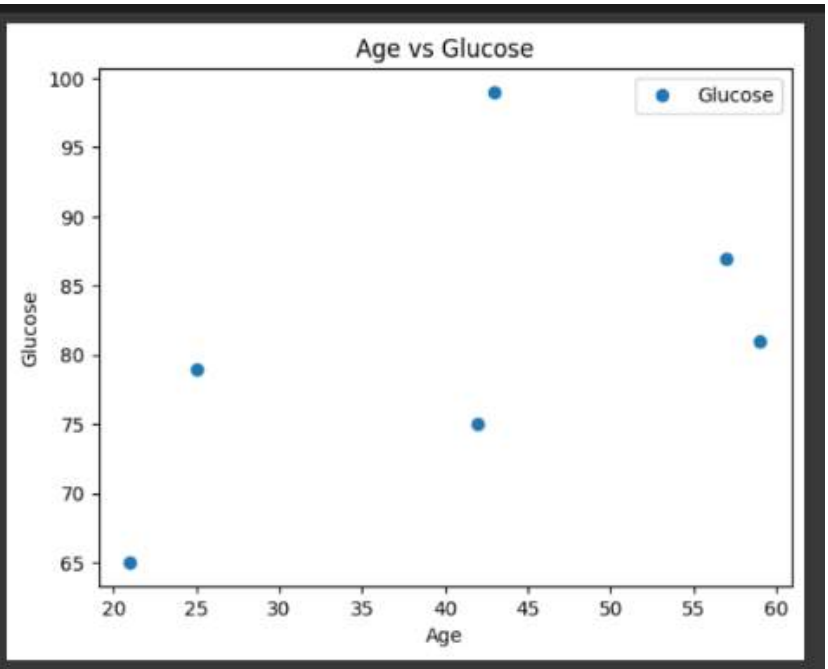
```
d = {'Age': [43, 21, 25, 42, 57, 59], 'Glucose': [99, 65, 79, 75, 87, 81]}
```

```
n = len(d['Age'])
print(n)
```

```
6
```

```
df = pd.DataFrame(data = d)
```

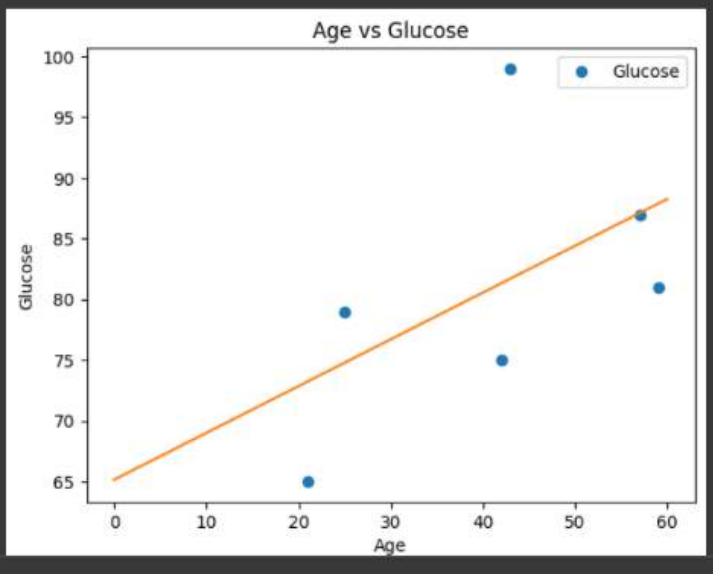
```
df.plot(x='Age', y='Glucose', style='o')
plt.xlabel('Age')
plt.ylabel('Glucose')
plt.title('Age vs Glucose')
plt.show()
```




```
xs=ys=xy=x=y=0
```

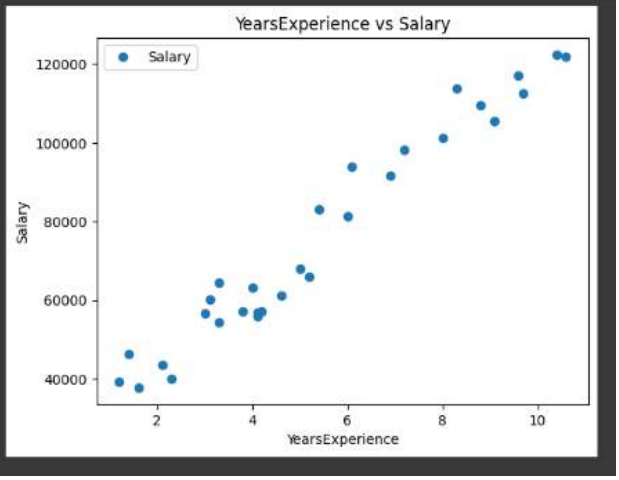
```
for i in range(0,n):
    x += d['Age'][i]
    y += d['Glucose'][i]
    xs += d['Age'][i]**2
    ys += d['Glucose'][i]**2
    xy += d['Age'][i]*d['Glucose'][i]
```

```
[ ] bo = ((y*xs)-(x*xy))/((n*xs)-(x**2))

[ ] b1 = ((n*xy)-(x*y))/((n*xs)-(x**2))

df.plot(x='Age', y='Glucose', style='o')
plt.plot([0, 60], [bo + b1*0, bo + b1*60])
plt.xlabel('Age')
plt.ylabel('Glucose')
plt.title('Age vs Glucose')
plt.show()
```



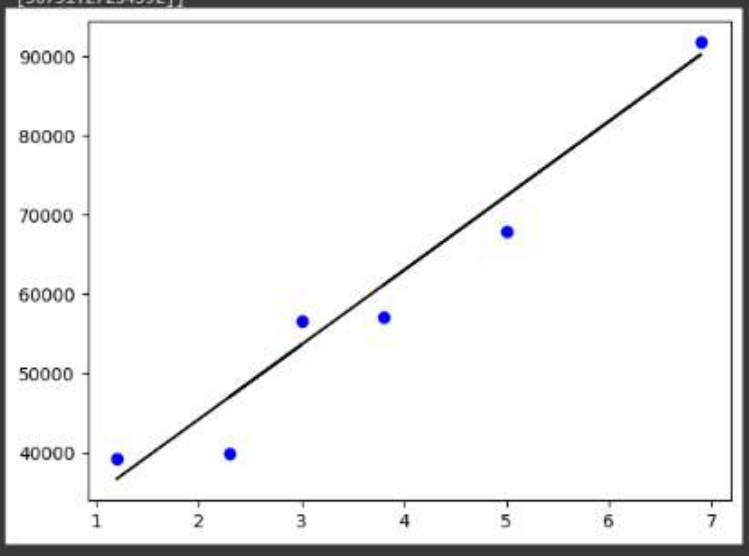
Then on an imported dataset using inbuilt functions and libraries

```
[ ] ycap = bo + (55*b1)
    print(ycap)

    86.32894560107455

[ ] df2 = pd.read_csv('Salary_dataset.csv')

df2.plot(x='YearsExperience', y='Salary', style='o')
plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.title('YearsExperience vs Salary')
plt.show()
```



```
[ ] from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    X = np.array(df2['YearsExperience']).reshape(-1, 1)
    y = np.array(df2['Salary']).reshape(-1, 1)

[ ] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

[ ] regr = LinearRegression()
    regr.fit(X_train, y_train)
    print(regr.score(X_test, y_test))

    0.9450507958680934

    y_pred = regr.predict(X_test)
    plt.scatter(X_test, y_test, color ='b')
    plt.plot(X_test, y_pred, color ='k')
    print(X_test)
    print(y_pred)
    plt.show()
```

```
    print(y_pred)
    plt.show()

    [[3.8]
     [6.9]
     [5. ]
     [2.3]
     [3. ]
     [1.2]]
    [[61140.05516025]
     [90242.83467203]
     [72405.64722933]
     [47058.0650739 ]
     [53629.66044753]
     [36731.27234392]]
```



And then the same procedure is implemented from scratch:

60004210210

Amartya Mishra

COMPS – C31

ML Experiment 2

60004210210
Amartya Mishra
COMPS - C31

ML Experiment 2

Aim: To Implement & understand the basics of logistic Regression.

Theory:
Logestic Regression is a supervised ML Algorithm used for classification task where the goal is to predict the probability that an instance belongs to a given class or not
It is a Statistical Algo which analyze the relation between two factors

Types of logistic Regression:

→ Binomial
→ Multinomial
→ Ordinal

$z = wx + b$

x: Independent inputs
b: biase term (intercept)
w: weights

Sigmoid function:
The output (z) is mapped to get the probability

$$\sigma(z) = \frac{1}{1 - e^{-z}}$$

logistic regression Equation

$$P(x, b, w) = \frac{1}{1 + e^{-wx+b}}$$

Evaluating Model Methods:

- → Accuracy
- → Precision
- → Recall
- → F1 score

Conclusion: Thus, we implemented logistic Regression

Implementation:

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

```
data = pd.read_csv('/content/dataset.csv')
X = data.iloc[:, :1]
y = data.iloc[:, 1]
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42)

logr = linear_model.LogisticRegression()
logr.fit(X_train,y_train)
y_pred = logr.predict(X_test)
```

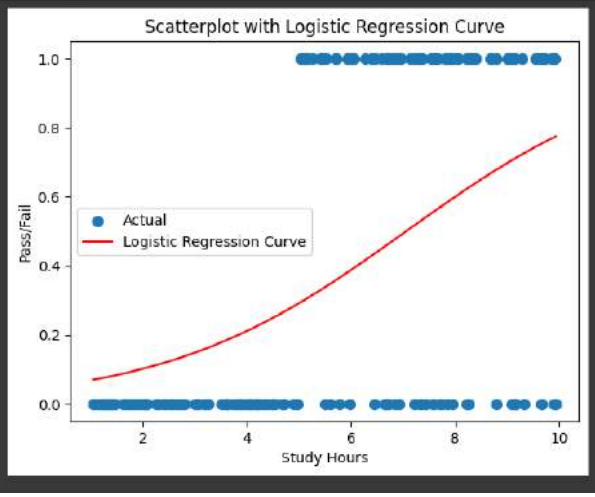
```
from sklearn import metrics

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
metrics.accuracy_score(y_test,y_pred)
```

```
0.74
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.linear_model import LogisticRegression

# Example data (assuming X_test and y_test are NumPy arrays)
# X_test = np.array([[10], [15], [20], [25]])
# y_test = np.array([0, 1, 1, 0])

# Check for DataFrame and correct if necessary
if isinstance(X_test, pd.DataFrame):
```

```
# Check for DataFrame and correct if necessary
if isinstance(X_test, pd.DataFrame):
    X_test = X_test.values.ravel()  # Assuming you want to use only the first column

# Convert labels to numerical representations if needed
le = LabelEncoder()
y_test = le.fit_transform(y_test)

# Normalize data (optional)
scaler = MinMaxScaler()
X_test_scaled = scaler.fit_transform(X_test.reshape(-1, 1))

# Fit logistic regression model
logr = LogisticRegression()
logr.fit(X_test_scaled, y_test)

# Create DataFrame
df = pd.DataFrame({'Study Hours': X_test.flatten(),
                   'Pass/Fail': y_test,
                   'Predicted': logr.predict(X_test_scaled)})

# Create scatterplot
df.plot.scatter(x='Study Hours', y='Pass/Fail', s=50, label='Actual')

# Generate x values for prediction
x_values = np.linspace(X_test.min(), X_test.max(), 100).reshape(-1, 1)

# Normalize x_values for prediction (if normalized earlier)
if 'scaler' in locals():
    x_values_scaled = scaler.transform(x_values)
    y_values = logr.predict_proba(x_values_scaled)[:, 1]
else:
    y_values = logr.predict_proba(x_values)[:, 1]

# Plot logistic regression curve
plt.plot(x_values.flatten(), y_values, color='red', label='Logistic Regression Curve')

# Set labels and title
plt.xlabel('Study Hours')
plt.ylabel('Pass/Fail')
```

```
# Plot logistic regression curve
plt.plot(x_values.flatten(), y_values, color='red', label='Logistic Regression Curve')

# Set labels and title
plt.xlabel('Study Hours')
plt.ylabel('Pass/Fail')
plt.title('Scatterplot with Logistic Regression Curve')

# Display legend
plt.legend()

# Show the plot
plt.show()
```




```
w0 = 1
w1 = 3
alpha = 0.001
```

```
def find_W(x, w0, w1):
  w = []
  for i in range(len(x)):
    w.append(w0 + w1*x[i])
  return w
```

```
def calculate_jw(x, w, y):
  temp = []
  for i in range(len(x)):
    temp.append((y[i]-1/(1+math.e**(-w[i]))))
  return sum(temp)
```

```
x=data['Study Hours']
y=data['Pass/Fail']

for i in range(125):
  w = find_W(x, w0, w1)
  jw = calculate_jw(x, w, y)
  prevW0=w0
  prevW1=w1
  w0 = w0 + alpha*(jw)
  w1 = w1 + alpha*(jw)
  if w0==prevW0 or w1==prevW1:
    break
```

```
pred=[]
for i in w:
  print(1/(1+math.e**-i),end=" ")
  if(1/(1+math.e**-i)>=0.5):
    pred.append(1)
  else:
    pred.append(0)
print()
```

```python
x=data['Study Hours']
y=data['Pass/Fail']

for i in range(125):
  w = find_W(x, w0, w1)
  jw = calculate_jw(x, w, y)
  prevW0=w0
  prevW1=w1
  w0 = w0 + alpha*(jw)
  w1 = w1 + alpha*(jw)
  if w0==prevW0 or w1==prevW1:
    break
```

```python
pred=[]
for i in w:
  print(1/(1+math.e**-i),end=" ")
  if(1/(1+math.e**-i)>=0.5):
    pred.append(1)
  else:
    pred.append(0)
print()
print(pred)
```

```
0.3038472332312542 0.5748126234529859 0.4681295531835178 0.4838898667875531 0.2213489056707168 0.2213487486231817 0.1990032396943072 0.5338568850270832 0.4058509159511583 0.4564623112840466 0.1789365386185 0.58399115108
[0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0,
```

```python
cnf_matrix = metrics.confusion_matrix(pred, y)
print(cnf_matrix)
metrics.accuracy_score(pred,y)*100
```

```
[[275 110]
 [ 41  66]]
68.2
```

60004210210

Amartya Mishra

COMPS – C31

ML Experiment 3

G0004210210
Amartya Mishra
COMPS - C31

ML Experiment - 3

Aim: To Implement CART Algorithm.

11/3/24

Theory:
CART (classification & Regression tree) is a variation of the decision tree Algorithm. It can handle both classification & Regression task. It is a predictive Algorithm used in ML & it Explains how the target variables values can be predicted based on other parameters

CART Algorithm

1) Tree Structure
2) Splitting criteria

$$gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

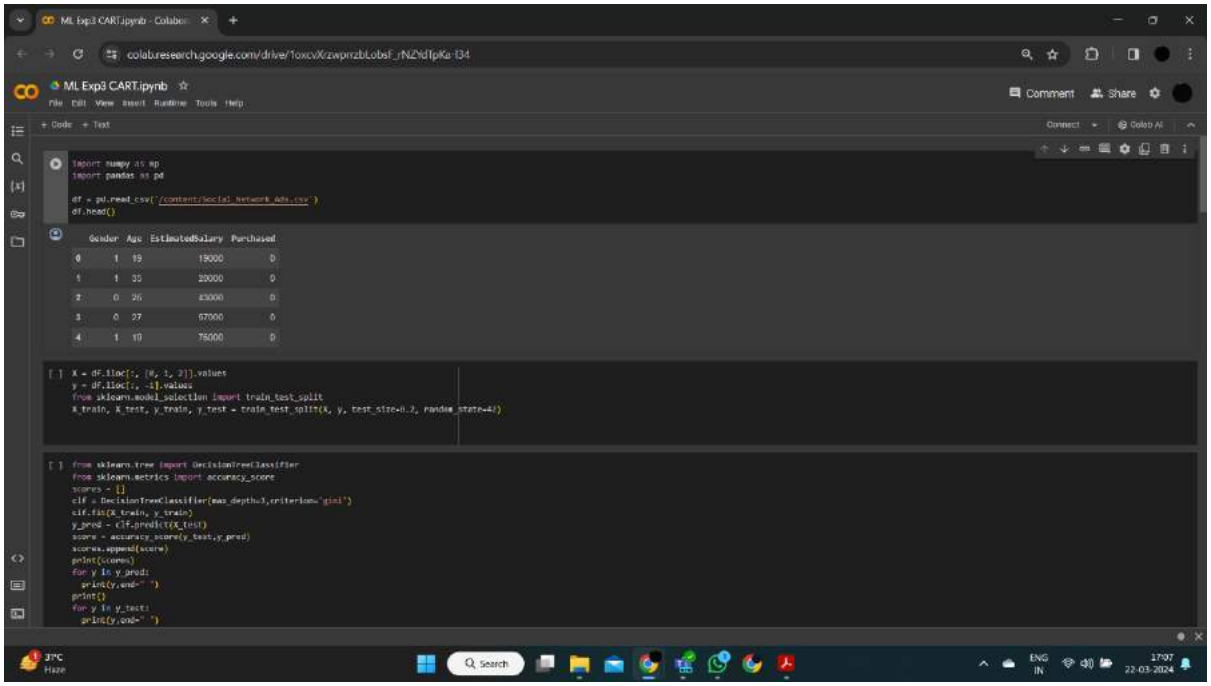
3) Pruning

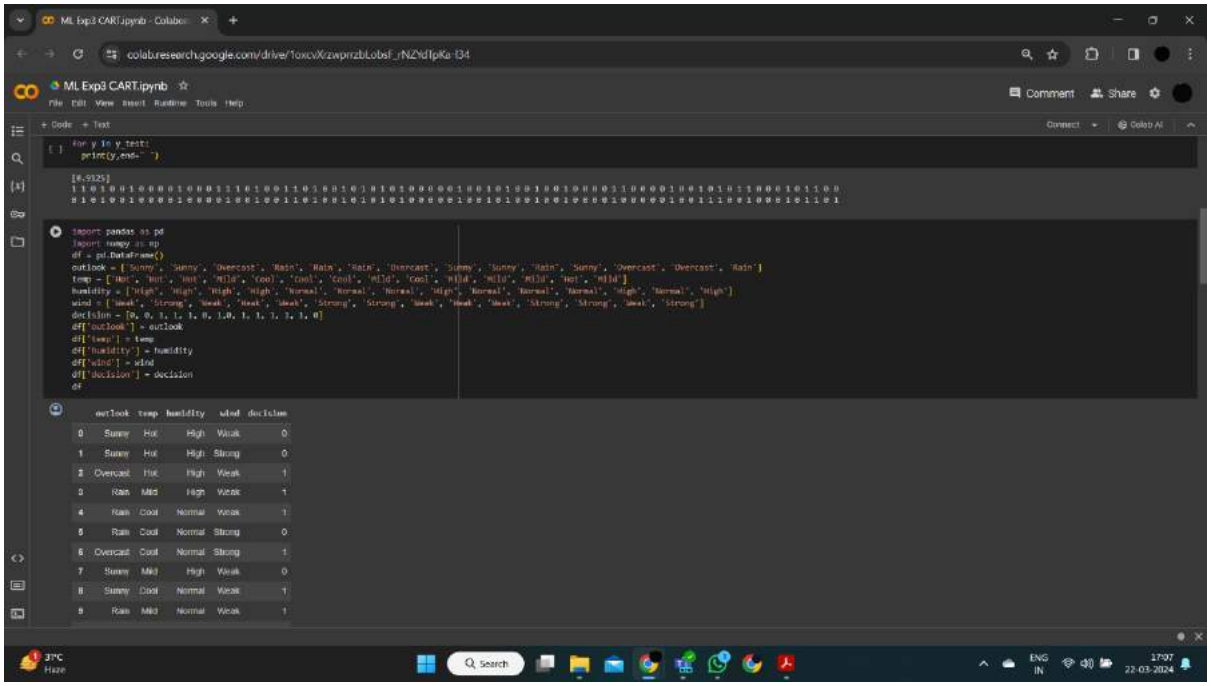
Advantages
1) Results are simplistic
2) Trees are non parametric & non-linear
3) Trees implicitly perform feature selection

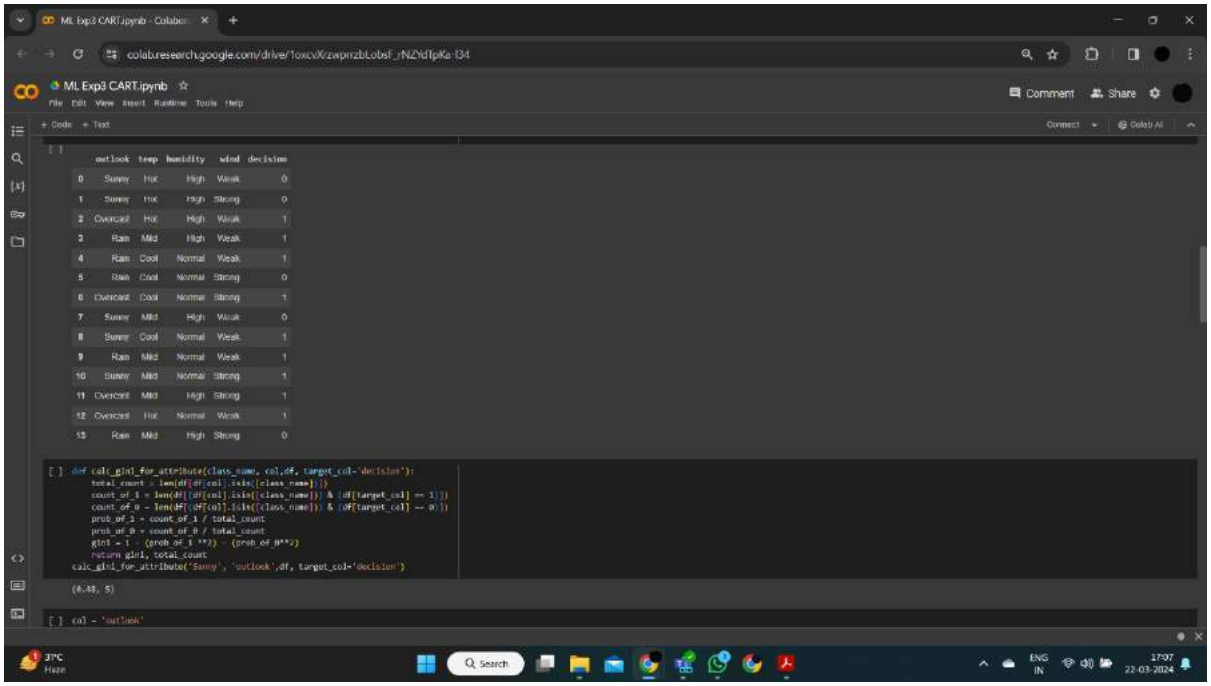
Disadvantages
1) overfitting
2) High variance
3) Low bias.

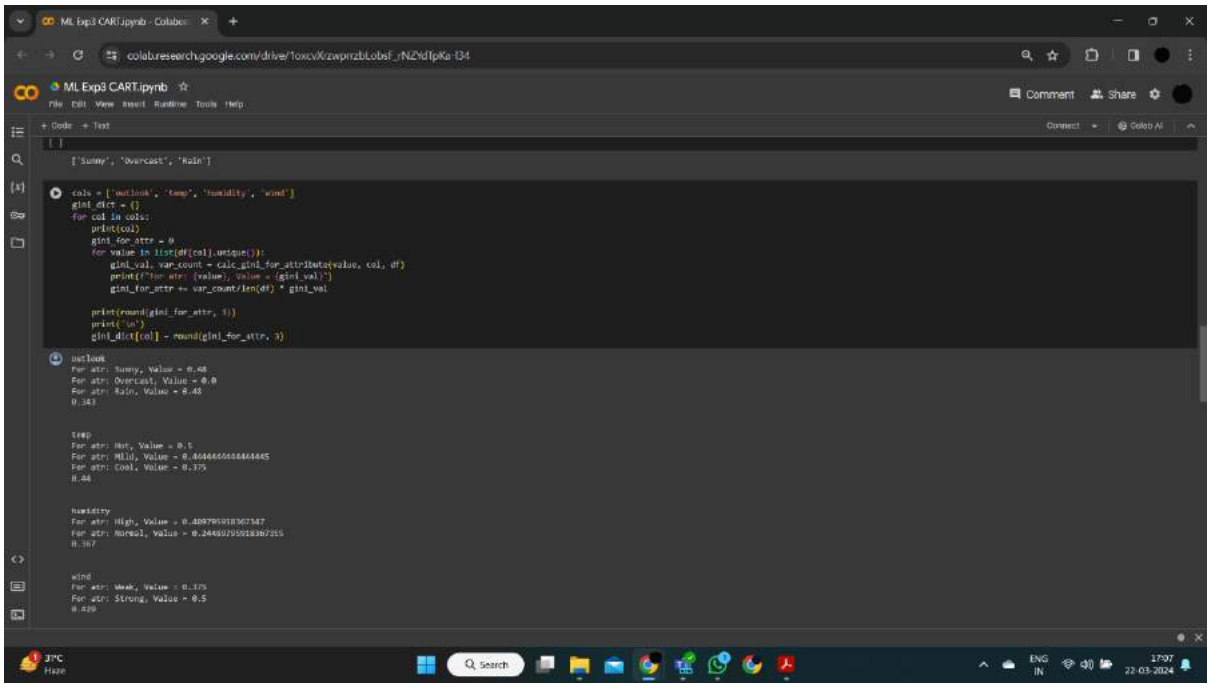
Conclusion: Thus we Implement CART Algorithm.

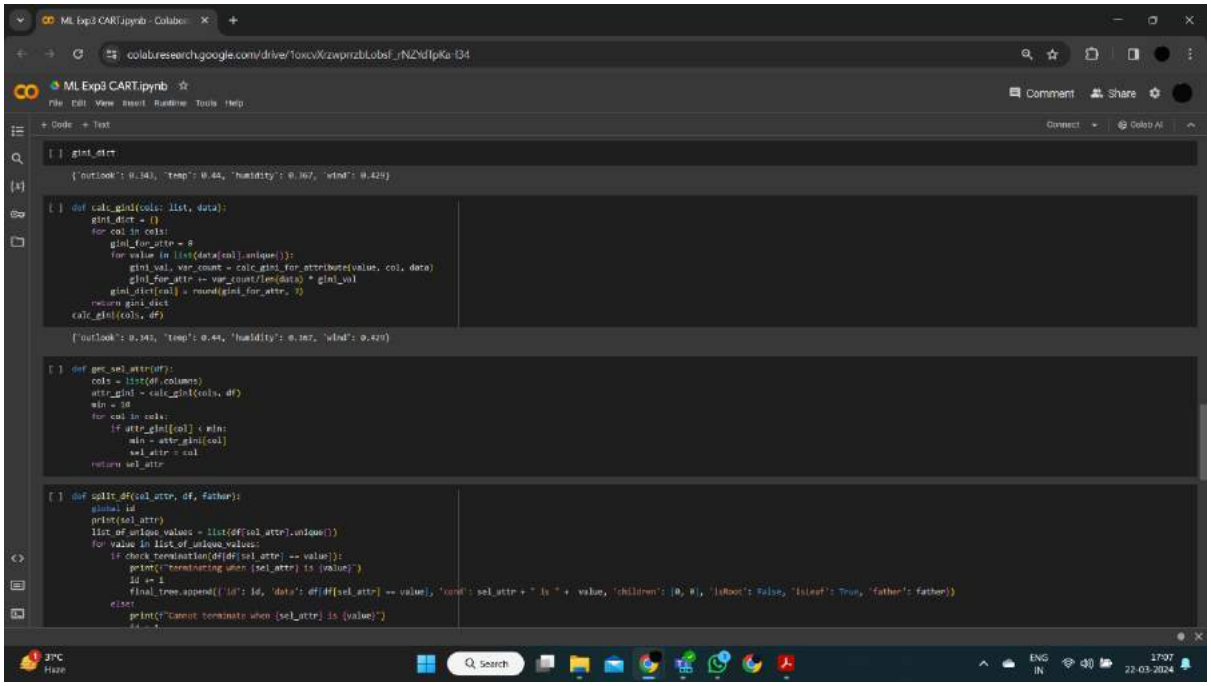
Implementation:

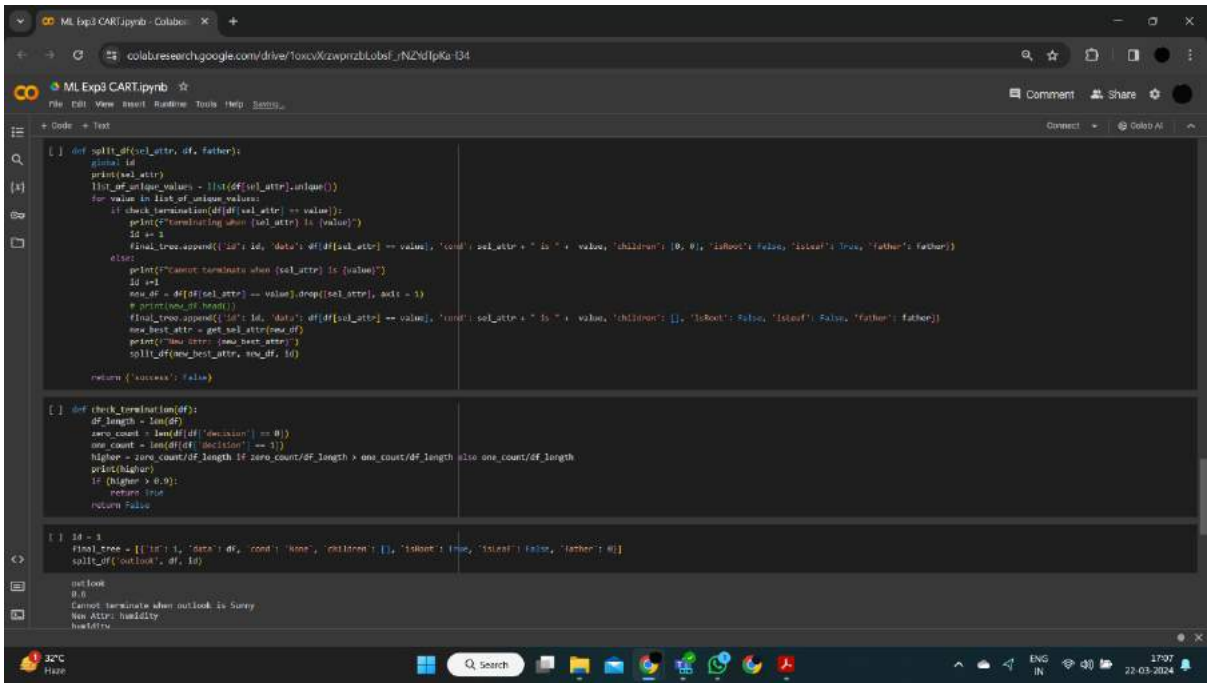


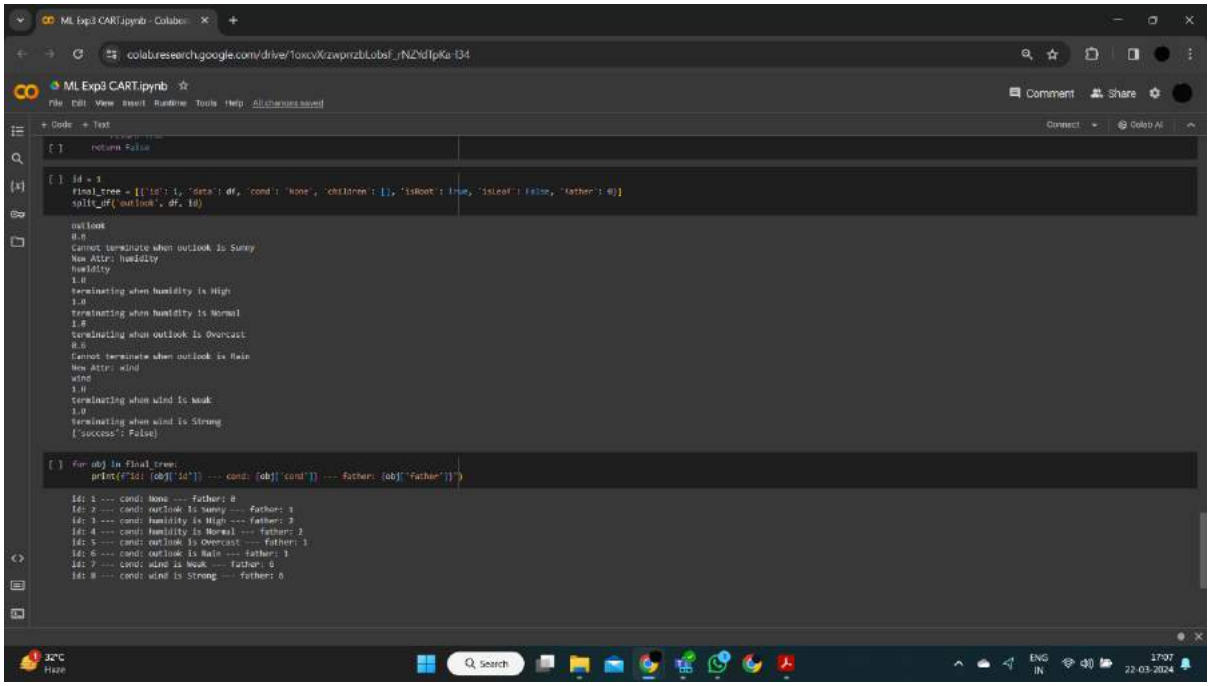



ML Exp3 CART.ipynb
colab.research.google.com/drive/1oxcvXzwpnzbLobsf_rNZYdTpKa-I34
ML Exp3 CART.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code + Text
Connect
Colab AI
outlook temp humidity wind decision
0 Sunny Hot High Weak 0
1 Sunny Hot High Strong 0
2 Overcast Hot High Weak 1
3 Rain Mild High Weak 1
4 Rain Cool Normal Weak 1
5 Rain Cool Normal Strong 0
6 Overcast Cool Normal Strong 1
7 Sunny Mild High Weak 0
8 Sunny Cool Normal Weak 1
9 Rain Mild Normal Weak 1
10 Sunny Mild Normal Strong 1
11 Overcast Mild High Strong 1
12 Overcast Hot Normal Weak 1
13 Rain Mild High Strong 0
def calc_gini_for_attribute(class_name, col,df, target_col='decision'):
total_count = len(df[df[col].isin([class_name])])
count_of_1 = len(df[(df[col].isin([class_name])) & (df[target_col] == 1)])
count_of_0 = len(df[(df[col].isin([class_name])) & (df[target_col] == 0)])
prob_of_1 = count_of_1 / total_count
prob_of_0 = count_of_0 / total_count
gini = 1 - (prob_of_1 **2) - (prob_of_0**2)
return gini, total_count
calc_gini_for_attribute('Sunny', 'outlook',df, target_col='decision')
(0.48, 5)
col = 'outlook'
37°C
Haze
Search
ENG
IN
17:07
22-03-2024


ML Exp3 CART.ipynb
colab.research.google.com/drive/1oxcvXzwpnzbLobsf_rNZYdTpKa-I34
ML Exp3 CART.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code + Text
Connect
Colab AI
['Sunny', 'Overcast', 'Rain']
cols = ['outlook', 'temp', 'humidity', 'wind']
gini_dict = {}
for col in cols:
print(col)
gini_for_attr = 0
for value in list(df[col].unique()):
gini_val, var_count = calc_gini_for_attribute(value, col, df)
print(f"For attr: {value}, Value = {gini_val}")
gini_for_attr += var_count/len(df) * gini_val
print(round(gini_for_attr, 3))
print('\n')
gini_dict[col] = round(gini_for_attr, 3)
outlook
For attr: Sunny, Value = 0.48
For attr: Overcast, Value = 0.0
For attr: Rain, Value = 0.48
0.343
temp
For attr: Hot, Value = 0.5
For attr: Mild, Value = 0.4444444444444445
For attr: Cool, Value = 0.375
0.44
humidity
For attr: High, Value = 0.4897959183673467
For attr: Normal, Value = 0.2448979591836735
0.367
wind
For attr: Weak, Value = 0.375
For attr: Strong, Value = 0.5
0.429
37°C
Haze
Search
ENG
IN
17:07
22-03-2024


ML Exp3 CART.ipynb
colab.research.google.com/drive/1oxcvXzwprrzbLobsF_rNZYdTpKa-I34
Comment
Share
+ Code + Text
Connect
Colab AI
gini_dict
{'Outlook': 0.343, 'Temp': 0.44, 'Humidity': 0.367, 'Wind': 0.429}
def calc_gini(cols: list, data):
gini_dict = {}
for col in cols:
gini_for_attr = 0
for value in list(data[col].unique()):
gini_val, var_count = calc_gini_for_attribute(value, col, data)
gini_for_attr += var_count/len(data) * gini_val
gini_dict[col] = round(gini_for_attr, 3)
return gini_dict
calc_gini(cols, df)
{'Outlook': 0.343, 'Temp': 0.44, 'Humidity': 0.367, 'Wind': 0.429}
def get_sel_attr(df):
cols = list(df.columns)
attr_gini = calc_gini(cols, df)
min = 10
for col in cols:
if attr_gini[col] < min:
min = attr_gini[col]
sel_attr = col
return sel_attr
def split_df(sel_attr, df, father):
global id
print(sel_attr)
list_of_unique_values = list(df[sel_attr].unique())
for value in list_of_unique_values:
if check_termination(df[df[sel_attr] == value]):
print(f"terminating when {sel_attr} is {value}")
id += 1
final_tree.append({'id': id, 'data': df[df[sel_attr] == value], 'cond': sel_attr + " is " + value, 'children': [0, 0], 'isRoot': False, 'isLeaf': True, 'father': father})
else:
print(f"Cannot terminate when {sel_attr} is {value}")
37°C
Haze
Search
ENG
IN
17:07
22-03-2024


ML Exp3 CART.ipynb
colab.research.google.com/drive/1oxcvXzwprrzbLobsF_rNZYdTpKa-I34
Comment
Share
Saving...
+ Code + Text
Connect
Colab AI
def split_df(sel_attr, df, father):
global id
print(sel_attr)
list_of_unique_values = list(df[sel_attr].unique())
for value in list_of_unique_values:
if check_termination(df[df[sel_attr] == value]):
print(f"terminating when {sel_attr} is {value}")
id += 1
final_tree.append({'id': id, 'data': df[df[sel_attr] == value], 'cond': sel_attr + " is " + value, 'children': [0, 0], 'isRoot': False, 'isLeaf': True, 'father': father})
else:
print(f"Cannot terminate when {sel_attr} is {value}")
id +=1
new_df = df[df[sel_attr] == value].drop([sel_attr], axis = 1)
# print(new_df.head())
final_tree.append({'id': id, 'data': df[df[sel_attr] == value], 'cond': sel_attr + " is " + value, 'children': [], 'isRoot': False, 'isLeaf': False, 'father': father})
new_best_attr = get_sel_attr(new_df)
print(f'New Attr: {new_best_attr}')
split_df(new_best_attr, new_df, id)
return {'success': False}
def check_termination(df):
df_length = len(df)
zero_count = len(df[df['decision'] == 0])
one_count = len(df[df['decition'] == 1])
higher = zero_count/df_length if zero_count/df_length > one_count/df_length else one_count/df_length
print(higher)
if (higher > 0.9):
return True
return False
id = 1
final_tree = [{'id': 1, 'data': df, 'cond': 'None', 'children': [], 'isRoot': True, 'isLeaf': False, 'father': 0}]
split_df('Outlook', df, id)
Outlook
0.6
Cannot terminate when Outlook is Sunny
New Attr: Humidity
Humidity
32°C
Haze
Search
ENG
IN
17:07
22-03-2024


ML Exp3 CART.ipynb - Colabo
colab.research.google.com/drive/1oxcvXrzwpnzbLobsf_rNZYdTpKa-I34
ML Exp3 CART.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Connect
Colab AI
return False
id = 1
final_tree = [{'id': 1, 'data': df, 'cond': 'None', 'children': [], 'isRoot': True, 'isLeaf': False, 'father': 0}]
split_df('outlook', df, 1d)
outlook
0.0
Cannot terminate when outlook is Sunny
New Attr: humidity
humidity
1.0
terminating when humidity is High
1.0
terminating when humidity is Normal
1.0
terminating when outlook is Overcast
0.0
Cannot terminate when outlook is Rain
New Attr: wind
wind
1.0
terminating when wind is Weak
1.0
terminating when wind is Strong
{'success': False}
for obj in final_tree:
print(f"id: {obj['id']} --- cond: {obj['cond']} --- father: {obj['father']}")
id: 1 --- cond: None --- father: 0
id: 2 --- cond: outlook is Sunny --- father: 1
id: 3 --- cond: humidity is High --- father: 2
id: 4 --- cond: humidity is Normal --- father: 2
id: 5 --- cond: outlook is Overcast --- father: 1
id: 6 --- cond: outlook is Rain --- father: 1
id: 7 --- cond: wind is Weak --- father: 6
id: 8 --- cond: wind is Strong --- father: 6
32°C
Haze
Search
ENG
IN
17:07
22-03-2024

60004210210

Amartya Mishra

COMPS – C31

ML Experiment 4

60042102l0
Amartya Mishra
COMPS - C31

ML Experiment 4

Aim: To implement principle component Analysis (PCA)

Theory:
PCA works on the condition that while the Data in a higher dimensional space is mapped to data in lower dimension space the Variance of the data in the ~~to~~ lower Dimensional Space should be maximum

PCA is a statistical procedure that uses an orthogonal transformation that converts a set of co-related variables to a set of uncorrelated variables
It is an unsupervised learning algorithm used to examine the iterations among a set of variables

Steps
I) Standardization

$$z = \frac{X - \mu}{\sigma}$$

II) Covariance Matrix Computation

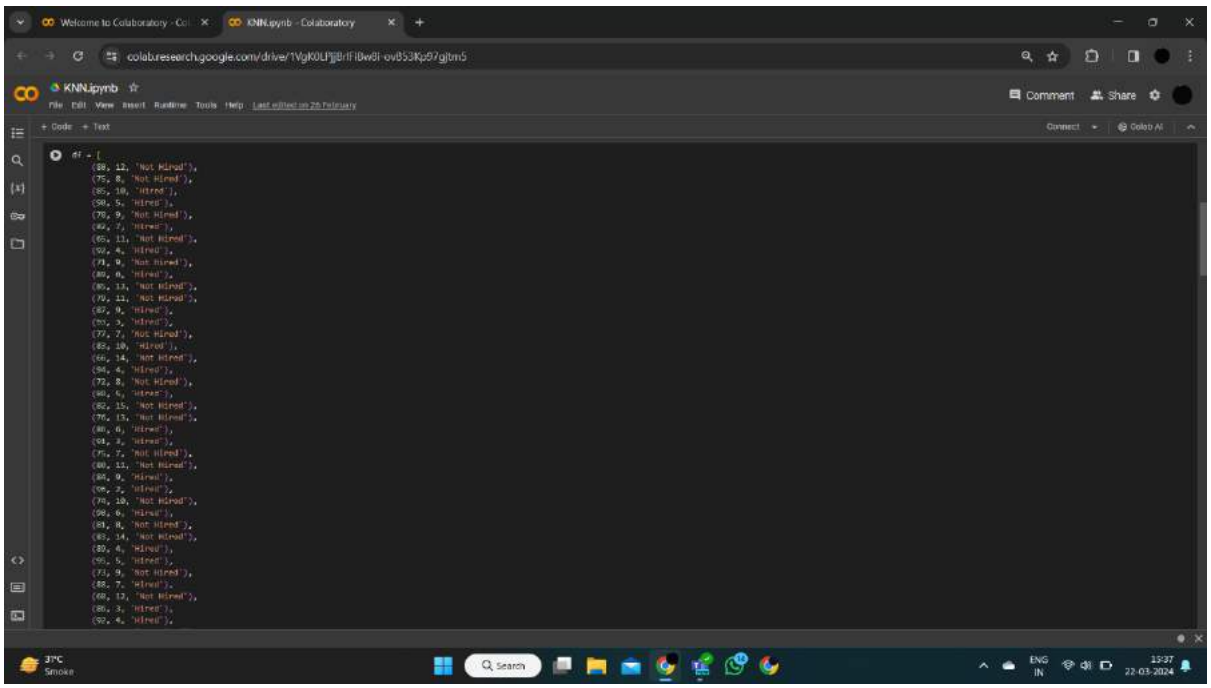
$$Cov(x_1, x_2) = \sum_{i=1}^{n} \frac{(x_i - \hat{x}_1)(x_{2i} - \bar{x}_2)}{n-1}$$

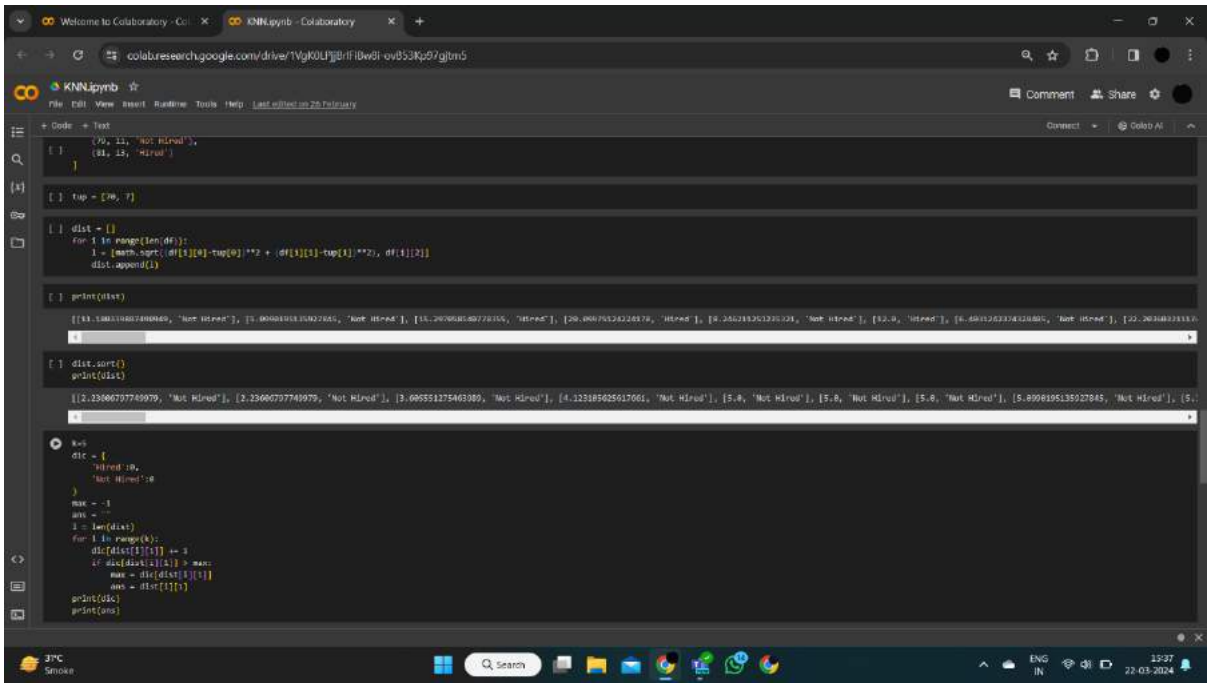
iii) Compute Eigen values & Eigen vectors of covariance matrix to identify principal components.

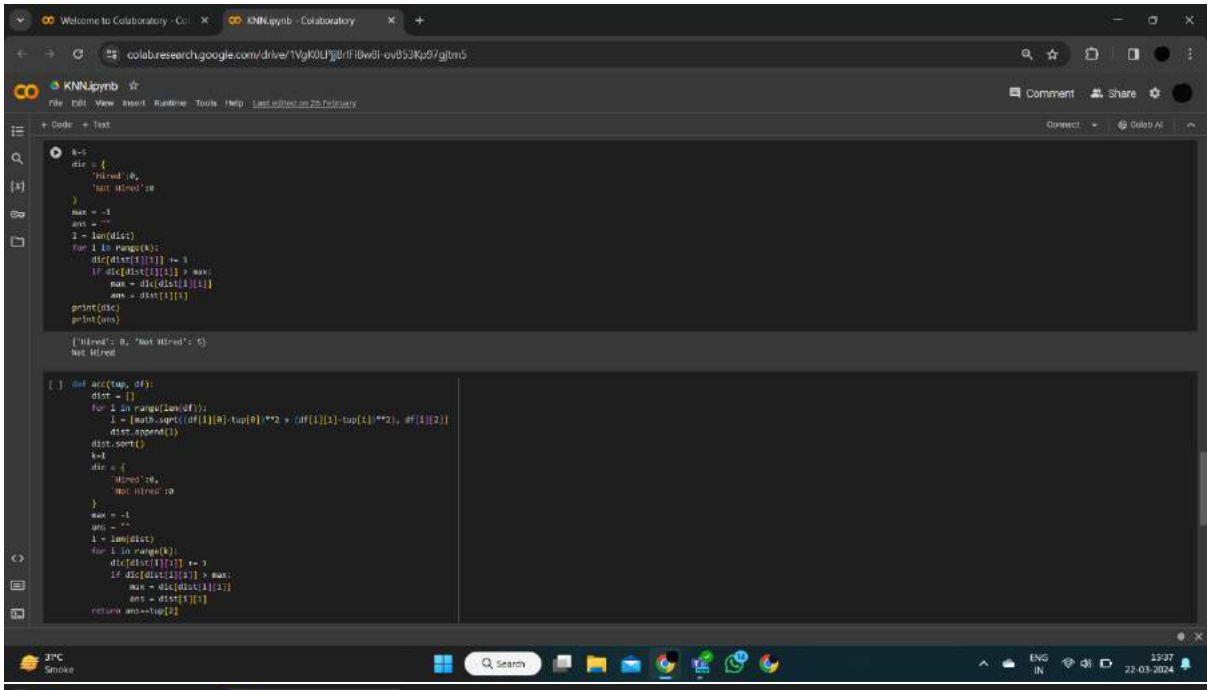
$$AX = \lambda X$$
$$AX - \lambda X = 0$$
$$(A - \lambda I)X = 0$$

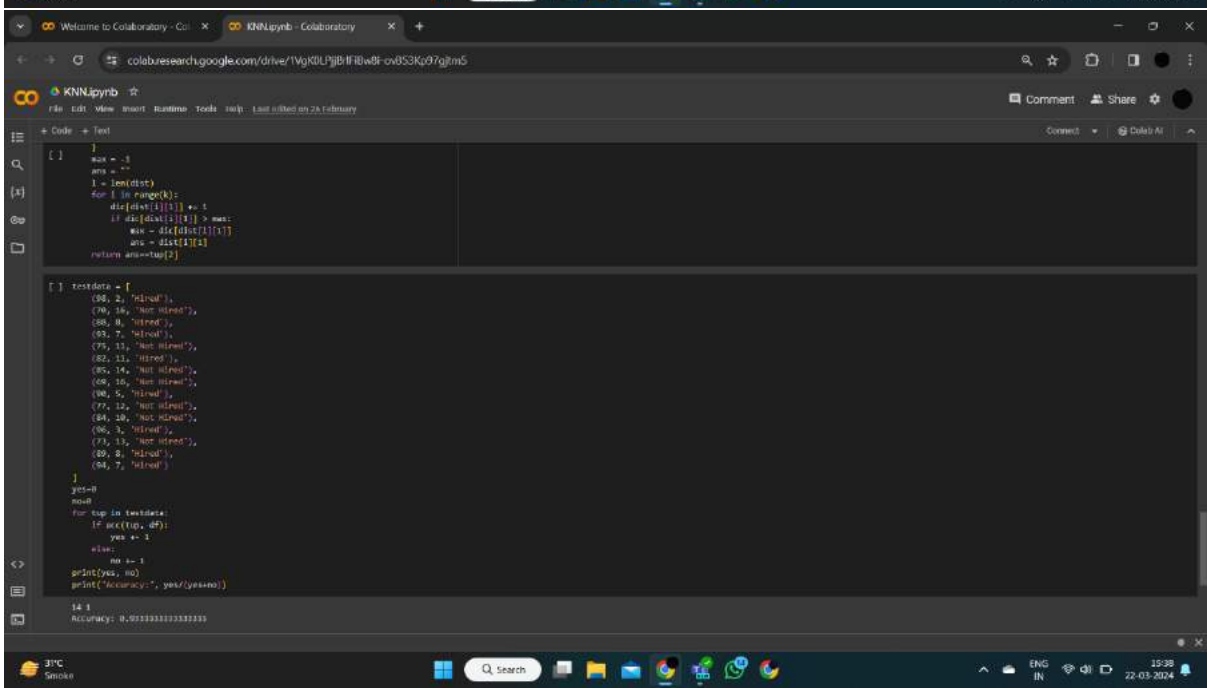
Conclusion : Thus we implement PCA

Implementation:






Welcome to Colaboratory - Co
KNN.ipynb - Colaboratory
colab.research.google.com/drive/1VgK0LPjjBrIFiBw8I-ov853Kp97gjtm5
KNN.ipynb
File Edit View Insert Runtime Tools Help Last edited on 26 February
Comment
Share
+ Code + Text
Connect
Colab AI
k=5
dic = {
'Hired':0,
'Not Hired':0
}
max = -1
ans = ""
l = len(dist)
for i in range(k):
dic[dist[i][1]] += 1
if dic[dist[i][1]] > max:
max = dic[dist[i][1]]
ans = dist[i][1]
print(dic)
print(ans)
{'Hired': 0, 'Not Hired': 5}
Not Hired
def acc(tup, df):
dist = []
for i in range(len(df)):
l = [math.sqrt((df[i][0]-tup[0])**2 + (df[i][1]-tup[1])**2), df[i][2]]
dist.append(l)
dist.sort()
k=5
dic = {
'Hired':0,
'Not Hired':0
}
max = -1
ans = ""
l = len(dist)
for i in range(k):
dic[dist[i][1]] += 1
if dic[dist[i][1]] > max:
max = dic[dist[i][1]]
ans = dist[i][1]
return ans==tup[2]
31°C
Smoke
Search
ENG
IN
15:37
22-03-2024


Welcome to Colaboratory - Co
KNN.ipynb - Colaboratory
colab.research.google.com/drive/1VgK0LPjjBrIFiBw8I-ov853Kp97gjtm5
KNN.ipynb
File Edit View Insert Runtime Tools Help Last edited on 26 February
Comment
Share
+ Code + Text
Connect
Colab AI
]
max = -1
ans = ""
l = len(dist)
for i in range(k):
dic[dist[i][1]] += 1
if dic[dist[i][1]] > max:
max = dic[dist[i][1]]
ans = dist[i][1]
return ans==tup[2]
testdata = [
(98, 2, 'Hired'),
(70, 16, 'Not Hired'),
(88, 8, 'Hired'),
(93, 7, 'Hired'),
(75, 11, 'Not Hired'),
(82, 11, 'Hired'),
(85, 14, 'Not Hired'),
(68, 16, 'Not Hired'),
(98, 5, 'Hired'),
(77, 12, 'Not Hired'),
(84, 10, 'Not Hired'),
(96, 3, 'Hired'),
(73, 13, 'Not Hired'),
(89, 8, 'Hired'),
(94, 7, 'Hired')
]
yes=0
no=0
for tup in testdata:
if acc(tup, df):
yes += 1
else:
no += 1
print(yes, no)
print("Accuracy:", yes/(yes+no))
14 1
Accuracy: 0.9333333333333333
31°C
Smoke
Search
ENG
IN
15:38
22-03-2024

60004210210

Amartya Mishra

COMPS – C31

ML Experiment 5

6000 4210210
Amartya Mishra
COMPS - C31

ML Experiment 5

Aim: To implement KNN Algorithm

Theory:
K-nearest Neighbor (KNN) Algo is a Supervised ML Algo employed to ~~take~~ tackle classification & Regression models problems.

Distance Metrics used in KNN

1) Euclidean dist

$$dist(x, y_i) = \sqrt{\sum_{j=1}^{d} (x_j - x_{ij})^2}$$

2) Manhattan Distance

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

3) Minkowski Distance

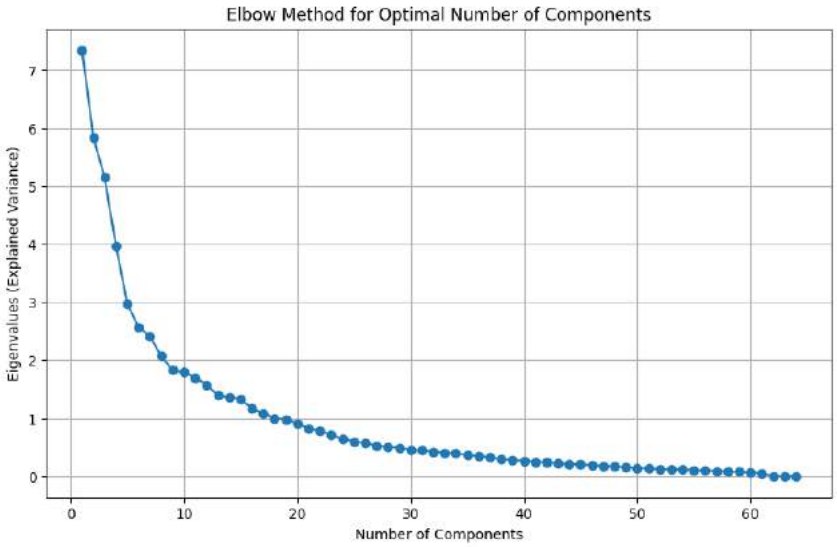
$$d(x, y) = \left(\sum_{i=1}^{n} (x_i - y_i)^p \right)^{1/p}$$

choosing the value of k
The value of k depends on input data
If input data has more outliers, a higher value of k is

Conclusion: Thus we implemented & understood KNN Algorithm.

Implementation:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
digits_data = load_digits()
X = digits_data.data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
eigenvalues = pca.explained_variance_
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(eigenvalues) + 1), eigenvalues, marker='o',
linestyle='-')
plt.title('Elbow Method for Optimal Number of Components')
plt.xlabel('Number of Components')
plt.ylabel('Eigenvalues (Explained Variance)')
plt.grid(True)
plt.show()
optimal_num_components = 10
X_reduced = X_pca[:, :optimal_num_components]
df_reduced = pd.DataFrame(X_reduced, columns=[f'PC{i}' for i in
range(1, optimal_num_components + 1)])
df_reduced['target'] = digits_data.target
df_reduced.to_csv('reduced_digits_dataset.csv', index=False)
print("Digits Wine dataset saved successfully.")
```




```
print(X_pca)
```

```
[[ 1.91421366e+00 -9.54501571e-01 -3.94603482e+00 ... -0.00000000e+00
   0.00000000e+00  8.24385469e-15]
 [ 5.88980330e-01  9.24635800e-01  3.92475494e+00 ...  8.37680823e-16
  -1.88056170e-17  3.08670512e-17]
 [ 1.30203906e+00 -3.17188827e-01  3.02333293e+00 ...  4.30699370e-16
   1.79508547e-17  2.15555254e-17]
```

60004210210

Amartya Mishra

COMPS – C31

ML Experiment 6

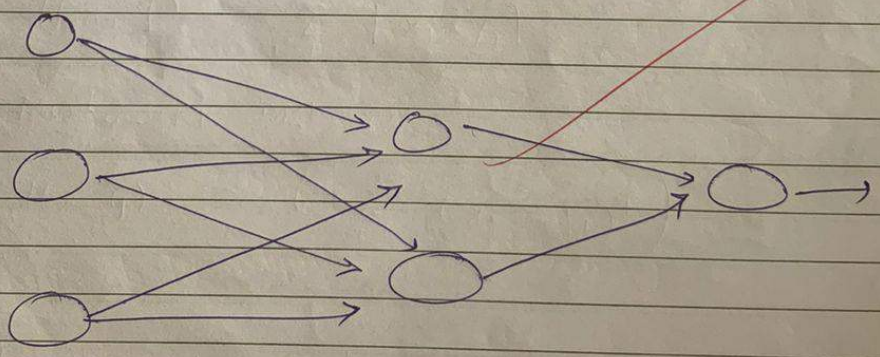
60004210210
Amartya Mishra
COMPS - C31

ML - Experiment 6

Aim: To implement Back propogation

Theory:
Back propogt^n is an algorithm that Back Propogates the errors from the output nodes to unput nodes
It is widely used Algorithm for training feed forward neural networks
It computes the gradient of loss function with respect to the network weights.

It is very efficient, rather than naively directly computing the gradient concerning each weight



Parameters:
x = inputs training vector $(x_1\ x_2\ \ldots\ x_n)$
t = target of vector $(t_1\ t_2\ \ldots\ t_6)$
δ_k = error at output unit

δ_j = error at hidden layer
α = learning rate
V_{oj} = biase of hidden unit j

conclusion: Thus, we implemented back Propogation

Implementation:

```
 import numpy as np
class NeuralNetwork:
def __init__(self, input_size, hidden_size, output_size):
self.input_size = input_size
self.hidden_size = hidden_size
self.output_size = output_size
# Initialize weights and biases
self.weights_input_hidden = np.random.randn(self.input_size,
self.hidden_size)
self.bias_input_hidden = np.random.randn(1, self.hidden_size)
self.weights_hidden_output = np.random.randn(self.hidden_size,
self.output_size)
self.bias_hidden_output = np.random.randn(1, self.output_size)
def sigmoid(self, x):
return 1 / (1 + np.exp(-x))
def sigmoid_derivative(self, x):
return x * (1 - x)
def forward(self, inputs):
self.hidden_input = np.dot(inputs, self.weights_input_hidden) +
self.bias_input_hidden
self.hidden_output = self.sigmoid(self.hidden_input)
self.final_input = np.dot(self.hidden_output,
self.weights_hidden_output) + self.bias_hidden_output
self.final_output = self.sigmoid(self.final_input)
return self.hidden_output, self.final_output
def backward(self, inputs, targets, learning_rate):
error = targets - self.final_output
delta_output = error * self.sigmoid_derivative(self.final_output)
delta_hidden = np.dot(delta_output, self.weights_hidden_output.T) *
self.sigmoid_derivative(self.hidden_output)
self.weights_hidden_output += np.dot(self.hidden_output.T,
delta_output) * learning_rate
self.bias_hidden_output += np.sum(delta_output, axis=0,
keepdims=True) * learning_rate
self.weights_input_hidden += np.dot(inputs.T, delta_hidden) *
learning_rate
self.bias_input_hidden += np.sum(delta_hidden, axis=0,
keepdims=True) * learning_rate
return error
def train(self, inputs, targets, learning_rate):
hidden_output, final_output = self.forward(inputs)
error = self.backward(inputs, targets, learning_rate)
print("Output of hidden layer:")
print(hidden_output)
print("Output of output layer:")

print(final_output)
print("Error found:")
print(error)
print("Updated weights after 1 iteration:")
print("Weights from input to hidden layer:")
print(self.weights_input_hidden)
print("Weights from hidden to output layer:")
print(self.weights_hidden_output)
```

```
dataset = pd.read_csv('reduced_digits_dataset.csv')
inputs = dataset.drop(columns=['target']).values
targets = dataset['target'].values.reshape(-1, 1)
input_size = inputs.shape[1]
output_size = len(np.unique(targets))
hidden_size = 3
nn = NeuralNetwork(input_size, hidden_size, output_size)
nn.train(inputs, targets, learning_rate=0.1)
```

60004210210

Amartya Mishra

COMPS – C31

ML Experiment 7

ML Experiment 7

60004210210
Amartya Mishra
COMPS - C31

Aim: To Implement Support Vector Machine:

Theory:
Support Vector machine (SVM) is used for linear/ non linear classification, regression & even outlier detection
It is Supervised ML Algorithm.
The main objective of the SVM is to find the optimal hyperplane in a N-dimensional space that can separate the Data points in different classes in the feature space
The hyper plane tries that the margin between the closest points of different classes should be as maximum as possible

The Dimension of the hyper plane depends upon the number of features

Mathematical Intution

$$w^T x + b = 0$$

$$di = \frac{w^T x + b}{||w||}$$

$$\hat{y} = \begin{cases} 1 & : w^T x + b \geq 0 \\ 0 & : w^T x + b < 0 \end{cases}$$

Types of SVM

1) Linear
2) Non linear

Advantages

1) Effective in high dimensional cases
2) Its memory is efficient as it uses a subset of training points in the decision function called support vectors
3) Different kernal functions can be specified for the decision function & its possible to specify custom kernals.

Conclusion : Thus we implemented SVM.

Implementation:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
data = load_breast_cancer()
X = data.data
y = data.target
X
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

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

60004210210

Amartya Mishra

COMPS – C31

ML Experiment 8

ML - Experiment 8

60004210210
Amartya Mishra
COMPS - C31

Aim: To Implement Baysian classification

Theory:
Byasian Classification is a probabilistic approach to learning & inference based on different views of what it means to learn from Data in which probability is used to represent uncertainity about the relation ship being learnt.

It is used to determine the Probability of hypothesis with a pair knowledge

It depends on conditional probability
The Bayes thm is $P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$

P(A|B) Probability of A given B happened (Posterior)
P(B|A) Prob. of B given A occurs (likelyhood)
P(A) prob of A prior prob. (Prob of hypothesis)
P(B) Prob of B ~~Posterior~~ marginal Prob. (Prob of evidence)

Conclusion: Thus we implement Byasian classification.

Implementation:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
data = load_breast_cancer()
X = data.data
y = data.target
X
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
classifier = SVC(kernel=' ')
classifier.fit(X_train, Y_train)
Y_pred = classifier.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
```

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

ML - Assignment 1

60004210210
Amartya Mishra
COMPS - C31

1) No - kmeans & gaussian Mixture Model (GMM) will generally not produce same cluster for a given dataset / data.

K - Means

It is a centroid based clustering Algorithm. It works by initially assigning a fixed No. of centroids at random location
Then it iteratively arranges the data point to the clustered centroid

Gaussian Mixture Model

GMM is probabilistic clustering model that assumes the data is generated by a massive gaussian distribution
It uses expectation minimization.

Hence K - means & GMM make different assumpt & hence produce different cluster appropriation.
The choice of Algorithm depends on the characteristics of the data.

2) Hidden Markov Model is a statistical model that is used to describe the probabilities relation ship between a sequence of observation & a sequence of hidden states.

Applications:

1) Speech recognition:
can model statistical properties of words or sentences to recogniz patterns.

2) Gesture Recognition:
used to model different gestures based on ~~obfe~~ observed movements.

3) Bio informatics
Protine structure prediction, sequence alignment & model various biological sequence.

4) Robotics & Autonomous System.
Localization & mapping

5) Independent Component Analysis
It is used to seperate mixed signals into their original independent components
It assumes that input is a combination of sources By finding the linear transformation that maximizes the statistical Independence of the components.

DATE:

ICA can be used to extract components
It leverages non gaussian nature of signals.

4) Deep Neural Network: They are NN based are composed of multiple layers of nodes which attemp to model high level abstraction in data.
It consist of I/P layer, multiple hidden layer & output layer.
Each set of layer contains nodes for computation

They are capable of learning features that can be used for various task.

60004210210
Amartya Mishra
COMPS - C31

Machine learning Assignment 2

A+

Q1.

a) Video Surveillance :

Application :

Machine learning can be used in video surveillance for various tasks such as object detection, activity recognition & anomaly detection.

It can help automatically detect & track objects of interest, identifying suspicious behaviour & alerting security personals in real time.

There many such more application in almost all fields of industry from face detection, motion detection to automatic tag detection etc.

Suitable ML Technique :

Convolution Neural Networks (CNN's) are commonly used for such task due to their ability to effectively analyze spatial features in image or frames of videos

b) Sentiment Analysis :

→ Application :

Sentiment Analysis involves determining the

Sentiments or opinion Expressed in text.

It could be extracted from:

- Social Media Post
- Product reviews
- customer feed backs

- Such technology can be used by businesses to understand customer opinions or analyze user feed back for product improvement.

Suitable Methods:

- NLP - Natural language processing models.
- RNN - Recurrent Neural Networks.
- Transformer Models : EBPTA.

(c) Image recognition :

Application:
It involves identifying & classifying objects or patterns with image.
It is used in various Applications

→ Medical diagnosis
→ Autonomous vehicles
→ facial Recognition.

Suitable ML Techniques:

Convolution Neural Network - CNN are most suitable techniques for image recognition tasks