**APPLIED ARTIFICIAL INTELLIGENCE**

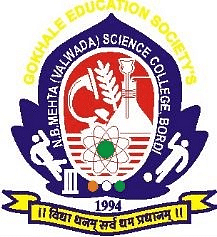
**MASTER OF SCIENCE**

**(INFORMATION TECHNOLOGY)**

**BY**

**Name: Mali Pranjal Krishna**

**Seat No: 2295009**



N.B. MEHTA (VALWADA) SCIENCE COLLEGE BORDI

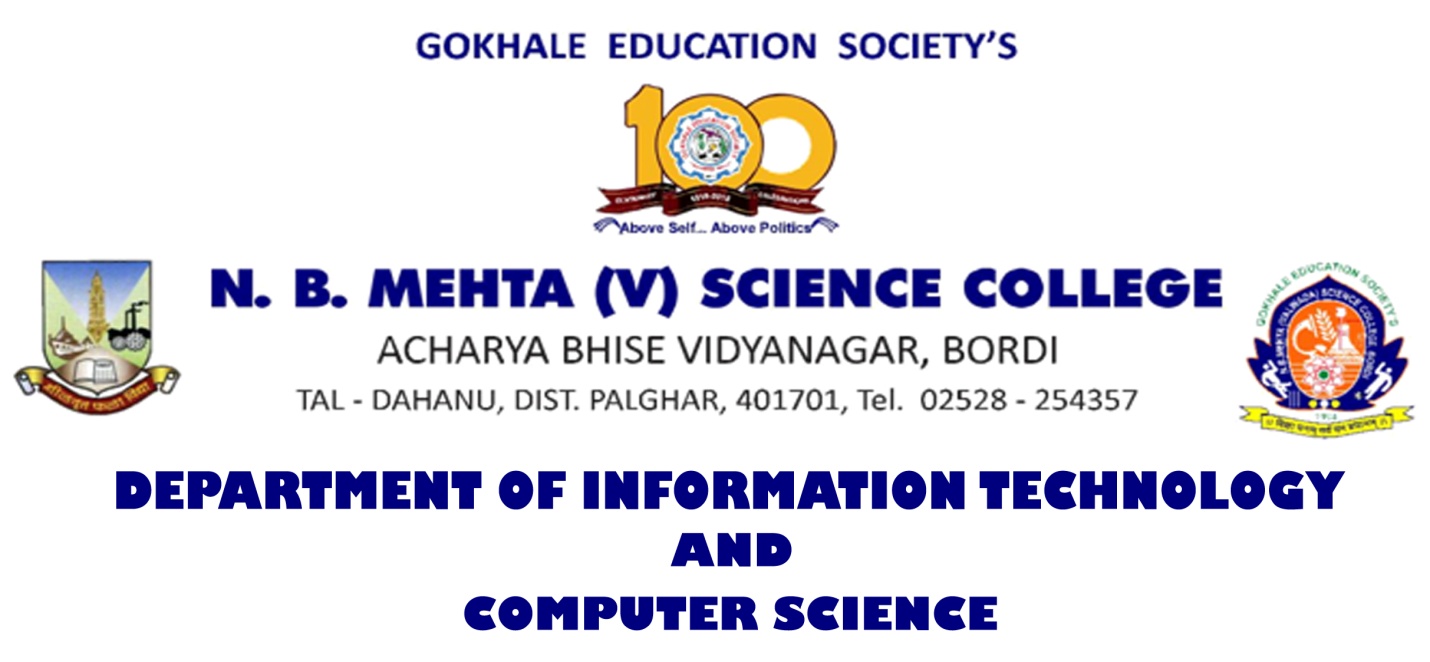
MAHARASHTRA - 401701

DEPARTMENT OF INFORMATION TECHNOLOGY

Master of Science (INFORMATION TECHNOLOGY)

Semester-III

Academic year 2023-2024



**Certificate**

Class: M.Sc. Information Technology (Semester III) Year: 2023-2024

This is to certify that the work entered in this journal is the work of

Kumari Prajapti Preeti Mangal Prasad of M.Sc. Part-1 division Information Technology Roll No. 10 Uni. Exam No. has satisfactorily completed the required number of practical and worked for the 1st term / 2nd term/ both the terms of the Year 2022-23 in the college laboratory as laid down by the university.

**Index**

|  |  |  |
| --- | --- | --- |
| Sr. No. | Practical Name | Sign |
| 1 | 1. Design a simple machine learning model to train the training instances and test the same using Python 2. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file |  |
| 2 | 1. Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking 2. For a given set of training data examples stored in .CSV file, implement and demonstrate the CandidateElimination algorithm to output a description of the set of all hypotheses consistent with the training examples |  |
| 3 | Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix |  |
| 4 | 1. For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm. (Use Univariate dataset) 2. For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm. (Use Multivariate dataset) |  |
| 5 | 1. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample 2. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set |  |
| 6 | 1. Implement the different Distance methods (Euclidean, Manhattan   Distance, Minkowski Distance) with Prediction, Test Score and Confusion Matrix   1. Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix |  |
| 7 | Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix |  |
| 8 | 1. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set 2. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs |  |

# Practical-01

A. Design a simple machine learning model to train the training instances and test the same using Python.

import numpy as np

class SVM: def init\_(self, learning\_rate=0.001, lambda\_param=0.01, n\_iters=1000):

self.lr = learning\_rate self.lambda\_param = lambda\_param self.n\_iters = n\_iters self.w = None

self.b - None

def fit(self, x, y): n\_samples, n\_features - X. shape

y\_ - np.where(y <- 0, -1, 1)

self.w = np.zeros(n\_features) self.b = 0

for \_in range(self.n\_iters): for idx, x\_i in enumerate(X): condition = y\_[idx] \* (np.dot(x\_i, self.w) - self.b) >= 1 if condition: self.w -= self.lr \* (2 " self.lambda\_param \* self.w) def predict(self, x): approx = np.dot(X, self.w) - self.b return np.sign(approx)

# Testing if \_name # Imports from sklearn import datasets

import matplotlib.pyplot as plt

x, y = datasets.make\_blobs(

n\_samples=50, n\_features=2, centers=2, cluster\_std=1.05, random\_state=40

y = np.where(y == 0, -1, 1)

clf = SVM() clf.fit(x, y)

# predictions - clf.predict(x)

" main

)

print(clf.w, clf.b)

def visualize\_svm():

def get\_hyperplane\_value(x, w, b, offset): return (-w[0] \* x + b + offset) / w[1]

fig = plt.figure() ax = fig.add\_subplot(1, 1, 1)

plt.scatter(x[:, 0], x[:, 1], marker-"o", c-y)

x0\_1 = np.amin(X[:, 0])

x0\_2 = np.amax(X[:, 0])

x1\_1 - get\_hyperplane\_value(x0\_1, clf.w, clf.b, 0) x1\_2 - get\_hyperplane\_value(x0\_2, clf.w, clf.b, 0)

x1\_1\_m = get\_hyperplane\_value(x0\_1, clf.w, clf.b, -1) x1\_2\_m = get\_hyperplane\_value(x0\_2, clf.w, clf.b, -1)

x1\_1\_p - get\_hyperplane\_value(x0\_1, clf.w, clf.b, 1)

x1\_2\_p = get\_hyperplane\_value(x0\_2, clf.w, clf.b, 1)

ax.plot([x0\_1, x0\_2], [x1\_1, x1\_2], "y -- ") ax.plot([x0\_1, x0\_2], [x1\_1\_m, x1\_2\_m], "k") ax.plot([x0\_1, x0\_2], [x1\_1\_p, x1\_2\_p], "k") x0\_1 - np.amin(X[:, 0])

x0\_2 = np.amax(x[:, 0])

x1\_1 = get\_hyperplane\_value(x0\_1, clf.w, clf.b, 0) x1\_2 = get\_hyperplane\_value(x0\_2, clf.w, clf.b, 0)

x1\_1\_m = get\_hyperplane\_value(x0\_1, clf.w, clf.b, -1) x1\_2\_m = get\_hyperplane\_value(x0\_2, clf.w, clf.b, -1)

x1\_1\_p =get\_hyperplane\_value(x0\_1, clf.w, clf.b, 1) x1\_2\_p - get\_hyperplane\_value(x0\_2, clf.w, clf.b, 1)

ax.plot([x0\_1, x0\_2], [x1\_1, x1\_2], "y -- ") ax.plot([x0\_1, x0\_2], [x1\_1\_m, x1\_2\_m], "k")

ax.plot([x0\_1, x0\_2], [x1\_1\_p, x1\_2\_p], "k")

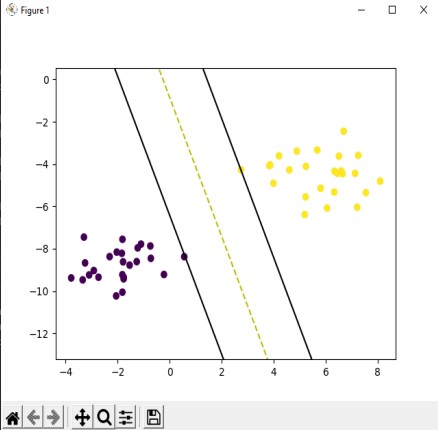
x1\_min = np.amin(X[:, 1]) x1\_max = np.amax(X[:, 1])

ax.set\_ylim([x1\_min - 3, x1\_max + 3])

plt. show()

visualize svm()

output:



B. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a

.CSV file.

Code:

import csv

num\_attributes = 5

a - [] print("\n") print("07\_Govind Saini")

print("\n The Given Training Data Set \n")

with open('govind.csv', 'r') as csvfile: reader = csv.reader(csvfile) for row in reader: a.append (row)

print(row)

print("\n The initial value of hypothesis: ") hypothesis = ['0'] \* num\_attributes print(hypothesis)

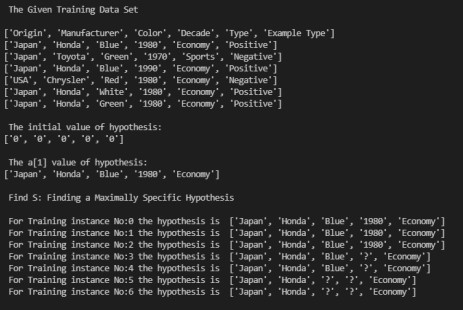
for j in range(0,num\_attributes): hypothesis[j] = a[1][j]

print("\n The a[1] value of hypothesis: ") print(hypothesis) print("\n Find S: Finding a Maximally Specific Hypothesis\n")

for i in range(e,len(a)): if a[i][num\_attributes] -- 'Positive': for j in range(0,num\_attributes): if a[i][j] != hypothesis[j]: hypothesis[j]='?' else :

hypothesis[j]= a[i][j] print(" For Training instance No:{} the hypothesis is ".format(i), hypothesis) print("\n The Maximally Specific Hypothesis for a given Training Examples :\n", hypothesis)

Output:



# Practical-02

A. Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.

Code: import numpy as np import matplotlib.pyplot as plt import pandas as pd

dataset = pd.read\_csv('winequality\_red.csv')

# distributing the dataset into two components X and Y X - dataset.iloc[:, 0:13].values

y = dataset.iloc[:, 13].values

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 = 0)

# performing preprocessing part

from sklearn.preprocessing import StandardScaler Sc = StandardScaler()

X\_train - sc.fit\_transforn(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.decomposition import PCA pca - PCA(n\_components - 2)

X\_train = pca.fit\_transform(X\_train) X\_test - pca.transform(X\_test)

explained variance = pca.explained variance ratio

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0) classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(x\_test)

from sklearn.metrics import confusion\_matrix

cm - confusion\_matrix(y\_test, y\_pred)

print("07\_Govind Saini")

from matplotlib.colors import ListedColormap

X\_set, y\_set - X\_train, y\_train

X1, X2 - np.meshgrid(np.arange(start - X\_set[:, @].min() - 1, stop - X\_set[:, 0].max() + 1, step - 0.01), np.arange(start = X\_set[:, 1].min() - 1,

stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(X1.shape), alpha = 0.75, cmap = Listedcolormap(('yellow', 'white', 'aquamarine'))) plt.xlim(X1.min(), X1.max()) plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c - ListedColormap(('red', 'green', 'blue'))(i), label - j)

plt.title('Logistic Regression (Training set)') plt.xlabel('PC1') plt.ylabel('PC2') plt. legend()

plt. show()

# Visualising the Test set results through scatter plot

from matplotlib.colors import ListedColormap

X\_set, y\_set - X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, e].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01), np.arange(start = X\_set[:, 1].min() - 1,

stop = X\_set[:, 1].max() + 1, step = 0.01))

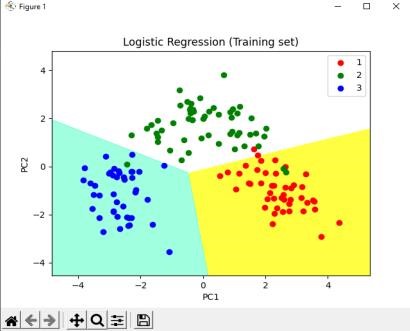
plt.contourf(X1, x2, classifier.predict(np.array([x1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha - 0.75, cmap = ListedColormap(('yellow', 'white', 'aquamarine'))) plt.xlim(X1.min(), X1.max()) plt.ylim(X2.min(), X2.max())

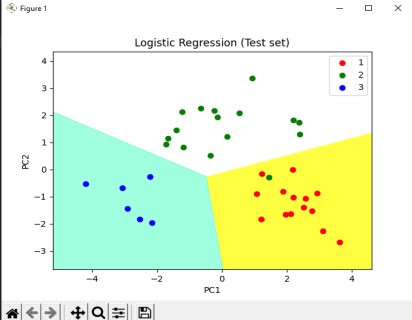
for i, j in enumerate(np.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set -= j, 1], C = ListedColormap(('red', 'green', 'blue') )(1), label = j)

plt.title('Logistic Regression (Test set)') plt.xlabel('PC1') plt.ylabel('PC2') plt. legend()

plt.show()

Output:





B. For a given set of training data examples stored in .CSV file, implement and demonstrate the CandidateElimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Code:

import csv import numpy as np import warnings warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt

def g\_e(n): return ("?",)n

def s\_0(n): return ('0'.)\*n

def more\_general(h1, h2): more\_general\_parts = [] for x, y in zip(h1, h2): mg - x -= "?" or (x !- "e" and (x -- y or y -= "e")) more\_general\_parts.append(mg)

return all(more\_general\_parts)

11 = [1, 2, 3] 12 = [3, 4, 5]

list(zip(11, 12))

# min\_generalizations def fulfills(example, hypothesis):

### the implementation is the same as for hypotheses:

return more\_general(hypothesis, example) def min\_generalizations(h, x): h\_new = list(h) for i in range(len(h)): if not fulfills(x[i:i+1], h[i:i+1]): h\_new[i] = '?' if h[i] 1= '0' else x[i] return [tuple(h\_new) ] min\_generalizations(h-('e', 'e'

x=('rainy', 'windy', 'cloudy'))

def min\_specializations(h, domains, x): results - [] for i in range(len(h)):

if h[i] -- "?": for val in domains[i]: if x[i] != val:

h\_new = h[:i] + (val, ) + h[i+1:] results.append(h\_new) elif h[i] != "0": h\_new = h[:i] + ('0', ) + h[i+1:] results.append(h\_new) return results min\_specializations(h=('?', 'x',), domains-[['a', x-('b', 'x'))

with open('season\_dataset.csv') as csvFile: examples - [tuple(line) for line in csv.reader(csvFile)]

, 'sunny"),

'b', 'c'], ['x', 'y']l,

def get\_domains(examples): d - [set() for i in examples[0]] for x in examples: for i, xi in enumerate(x): d[i].add(xi)

return [list(sorted(x)) for x in d]

get\_domains(examples)

def candidate\_elimination(examples): domains - get\_domains(examples)[ :- 1]

G = set([g\_0(len(domains))]) S = set([s\_0(len(domains))]) i-℮

#print("\n G[{0]]:".format(i),G) #print("\n S[{0]]:".format(i),S) for xcx in examples:

1=1+1

x, cx - xcx[ :- 1], xcx[-1] # Splitting data into attributes and decisions if cx -= 'Y': # x is positive example

G - {g for g in G if fulfills(x, g)} S = generalize\_S(x, G, S) else: # x is negative example

5 m {s for s in 5 if not fulfills(x, s)} def generalize\_S(x, G, S): S\_prev = list(S) for s in S\_prev: if s not in S: continue if not fulfills(x, s):

S.remove(s)

Splus - min\_generalizations(s, x)

## keep only generalizations that have a counterpart in G s.update([h for h in Splus if any([more\_general(g,h) for g in G])])

## remove hypotheses less specific than any other in S

5.difference\_update([h for h in S if any([more\_general(h, h1) for h1 in S if h !- hi])])

return S def specialize\_G(x, domains, G, 5): G\_prev - list(G) for g in G\_prev: if g not in G: continue if fulfills(x, g):

G.remove(g)

Ginus = min specializations(g, domains, x) def specialize\_G(x, domains, G, 5): G\_prev = list(G) for g in G\_prev: if g not in G: continue if fulfills(x, g):

G. remove(g)

Gminus - min\_specializations(g, domains, x)

## keep only specializations that have a conuterpart in S G.update([h for h in Gminus if any([more\_general(h, s) for s in S])])

ftt remove hypotheses less general than any other in G

G.difference\_update([h for h in G if any([more\_general(g1, h)

for g1 in G if h !- g1])])

G, S = candidate\_elimination(examples) print("G[4] ="

print("s[4] -"

return G class HypothesisNode(object): def \_init\_(self, h, level-0, parents-None): self.h = h self.level - level if parents is None:

parents = [] self.parents = set(parents) def \_repr\_(self): return "HypothesisNode({], {), {])".format(self.h, self.level, self.parents)

def build\_hypothesis\_space(G, S): levels = [[HypothesisNode(x, 0) for x in G]] curlevel = 1

def next\_level(h, S):

for s in S: for i in range(len(h)): if h[i] == '?' and s[i] |= '?': yield h[:i] + (s[i], ) + h[i+1:]

nextLv1 - {} while True: for n in levels[-1]:

for hyp in next\_level(n.h, 5): if hyp in nextLvl: nextLvl[hyp].parents.add(n) else: pos = 0 ymin = 0.1

ymax = 0.9

for nodes, y in [(levels[0], ymin), (levels[-1], ymax)]: xvals = np.linspace(0, 1, len(nodes)) for x, n in zip(xvals, nodes):

pos[n.h] - [x, y]

pos = nx.layout.fruchterman\_reingold\_layout(g, pos-pos, fixed-pos.keys())

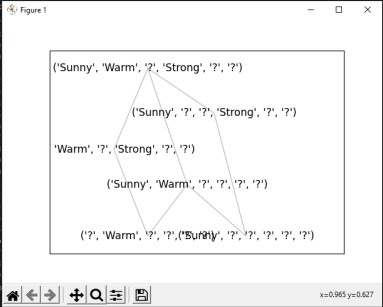
nx.draw\_networkx\_edges(g, pos-pos, alpha-0.25) nx.draw\_networkx\_labels(g, pos-pos)

plt.box(True) plt.xticks([]) plt.yticks([]) plt.xlim(-1, 2)

plt.gcf().set\_size\_inches((10, 10)) plt.show() print()

draw\_hypothesis\_space(G, 5)

Output:



# Practical-03

Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.

Code: import pandas as pd

import seaborn as sns

df = pd.read\_csv("HR\_employee\_data.csv") df.head()

sns.countplot(x='Attrition', data=df) df.drop(['EmployeeCount', 'EmployeeNumber', 'Over18', 'StandardHours'], axis-

"columns", inplace-True)

categorical\_col = [] for column in df.columns: if df[column].dtype == object and len(df[column].unique()) <= 50:

categorical col annand(column) (variable) df: DataFrame df['Attrition'] = df.Attrition.astype("category").cat.codes

from sklearn.preprocessing import LabelEncoder label = LabelEncoder() for column in categorical\_col:

df[column] = label.fit\_transform(df[column])

from sklearn.model\_selection import train\_test\_split

X = df.drop('Attrition', axis=1) y - df.Attrition

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report def print\_score(clf, X\_train, y\_train, X\_test, y\_test, train=True):

if train: pred - clf.predict(x\_train)

clf\_report = pd.DataFrame(classification\_report(y\_train, pred, output\_dict=True)) print("07\_Govind Saini \n") print("Train Result:\n ==

print(f"Accuracy Score: {accuracy\_score(y\_train, pred) \* 100 :. 2f}%") print(f"CLASSIFICATION REPORT:\n{clf\_report}")

print(f"Confusion Matrix: \n {confusion\_matrix(y\_train, pred)}\n")

elif train -- False:

pred - clf.predict(x\_test)

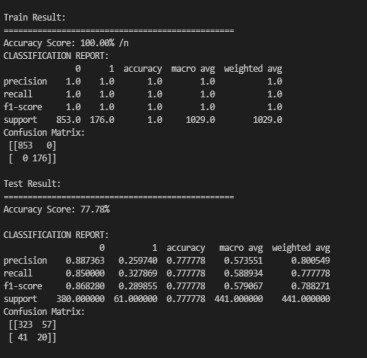
clf\_report = pd.DataFrame(classification\_report(y\_test, pred, output\_dict=True)) print("Test Result:\n ===== print(f"Accuracy Score: {accuracy\_score(y\_test, pred) \* 100 :. 2f}%")

print(f"CLASSIFICATION REPORT:\n{clf\_report}") print(f"Confusion Matrix: \n {confusion\_matrix(y\_test, pred)}\n")

from sklearn.tree import DecisionTreeClassifier tree\_clf = DecisionTreeClassifier(random\_state=42) tree\_clf.fit(x\_train, y\_train)

print\_score(tree\_clf, X\_train, y\_train, X\_test, y\_test, train=True) print\_score(tree\_clf, X\_train, y\_train, X\_test, y\_test, train=False)

Output:



# Practical-04

A. For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm. (Use Univariate dataset).

Code: import pandas as pd import numpy as np import matplotlib.pyplot as plt plt.rcparams['figure.figsize'] - (12.0, 9.0)

# Preprocessing Input data

data = pd.read\_csv('Sample\_Salary\_Data.csv')

X = data.iloc[:, 0] Y - data. iloc[:, 1] plt.scatter(x, Y)

plt. show()

# Building the model X\_mean - np.mean(X)

Y\_mean = np.mean(Y)

nun - 0 den = 8 for i in range(len(X)):

num +- (X[i] - x\_mean)\*(Y[i] - Y\_mean) den += (X[i] - X\_mean) \*\* 2 m - num / den C = Y\_mean - m\*x\_mean

print('07\_Govind Saini') print (m, c)

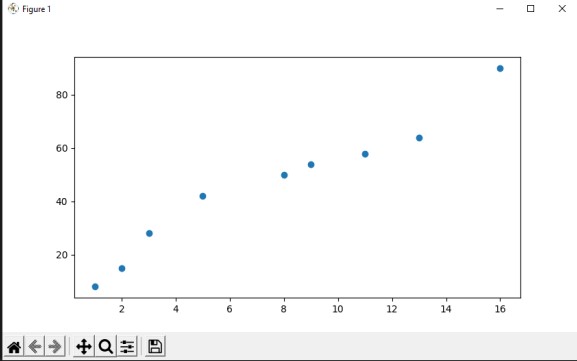
# Making predictions Y\_pred = m\*X + C

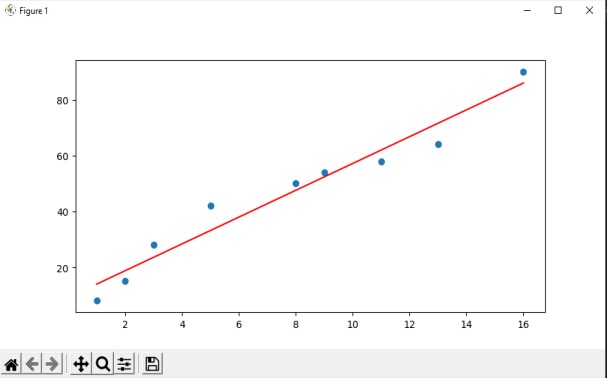
plt.scatter(X, Y) # actual

# plt.scatter(X, Y\_pred, color='red')

plt.plot([min(X), max(x)], [min(Y\_pred), max(Y\_pred)], color-'red') # predicted plt.show()

Output:





B. For a given set of training data examples stored in a .CSV file implement Logistic

Regression algorithm. (Use Multivariate dataset )

Code: import numpy as np import matplotlib.pyplot as plt import pandas as pd

dataset = pd.read\_csv('p4b.csv') X = dataset. iloc[:, [0, 1]].values

y = dataset.iloc[:, 2].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.25, random\_state = 0)

from sklearn.preprocessing import StandardScaler

5C = StandardScaler()

X\_train = sc.fit\_transform(X\_train) X\_test = sc.transform(X\_test)

print (X\_train[0:10, :])

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(x\_test)

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred)

print ("Confusion Matrix : \n", cm)

df - pd.DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

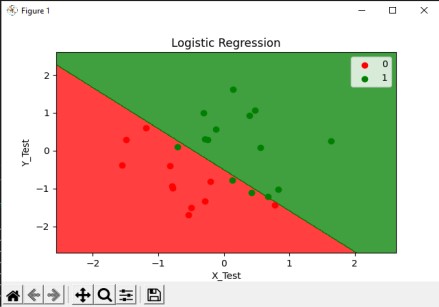
X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:,

0].max() + 1, step = 0.01), np.arange(start - X\_set[:, 1].min() - 1, stop - X\_set[:, 1].max() + 1, step - 0.01)) plt.contourf(X1, X2, classifier.predict(np.array([x1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green'))) plt.xlim(X1.min(), X1.max()) plt.ylim(X2.min(), X2.max()) for i, j in enumerate(np.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], x\_set[y\_set == j, 1], C = ListedColormap(('red', 'green'))(i), label = j) plt.title('Logistic Regression') plt.xlabel('X\_Test') plt.ylabel('Y\_Test') plt. legend()

plt. show()

Output:



# Practical-05

A. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Code: from chefboost import Chefboost as chet import pandas as pd

df - pd.read\_csv("Weekened\_data.txt") print("07\_Govind Saini") config = {'algorithm': 'ID3'}

model = chef.fit(df, config = config, target\_label = 'Decision')

prediction = chef.predict(model, param = ['W1', 'Sunny', 'Yes', 'Rich']) moduleName - "outputs/rules/rules" #this will load outputs/rules/rules.py tree = chef.restoreTree(moduleNane)

prediction = tree.findDecision(['W1', 'Sunny', 'Yes', 'Rich'])

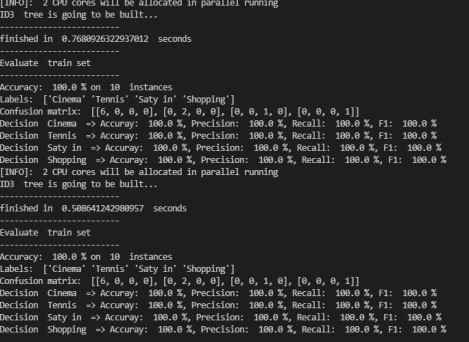
chef. save\_model(model, "model.pkl")

model - chef. load\_model("model.pk1")

prediction - chef.predict(model, ['W1',85,85, 'Rich'])

config = {'algorithm': 'ID3'} #Set algorithm to ID3, C4.5, CART, CHAID or Regression model - chef.fit(df, config)

Output:



B. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.

Code: from sklearn.neighbors import KNeighborsClassifier from sklearn.preprocessing import StandardScaler from sklearn.pipeline import make\_pipeline from sklearn import datasets from sklearn.model\_selection import train\_test\_split from sklearn.model\_selection import GridSearchCV

iris = datasets. load\_iris() X = iris.data

y = iris.target

# print(iris)

# Create train and test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=42, stratify-y)

# Feature Scaling using StandardScaler

SC = StandardScaler() sc.fit(x\_train)

X\_train\_std = sc.transform(x\_train)

X\_test\_std = sc.transform(X\_test)

# Fit the model knn = KNeighborsClassifier(

n\_neighbors=5, p=2, weights='uniform', algorithm='auto')

knn.fit(X\_train\_std, y\_train)

# Evaluate the training and test score print('07\_Govind Saini')

print('Training accuracy score: %.3f' % knn.score(X\_train\_std, y\_train)) print('Test accuracy score: %.3f' % knn.score(X\_test\_std, y\_test))

iris = datasets. load\_iris() X - iris.data

y - iris.target

# Create train and test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

x, y, test\_size=0.3, random\_state=42, stratify=y)

# Create a pipeline

pipeline = make\_pipeline(StandardScaler(), KNeighborsClassifier())

# Create the parameter grid

naram prid = If param\_grid = [{

'kneighborsclassifier\_n\_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 10],

'kneighborsclassifier\_p': [1, 2],

"kneighborsclassifier\_weights': ['uniform', 'distance'],

'kneighborsclassifier\_algorithm': ['auto', 'ball\_tree', 'kd\_tree', 'brute'],

# Create a grid search instance

gs = GridSearchcv(pipeline, param\_grid-param\_grid, scoring-'accuracy', refit-True,

CV=10, verbose=1,

n\_jobs=2)

# Fit the most optimal model

gs.fit(x\_train, y\_train)

# Print the best model parameters and scores print('Best Score: %.3f' % gs.best\_score\_, '\nBest Parameters: ', gs.best\_params\_)

Output:



# Practical-06

A. Implement the different Distance methods (Euclidean, Manhattan Distance, Minkowski Distance) with Prediction, Test Score and Confusion Matrix.

Code: from math import sqrt from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

# calculate euclidean distance def euclidean\_distance(a, b): return sqrt(sum((e1-e2) \*\* 2 for e1, e2 in zip(a, b)))

# calculate manhattan distance def manhattan\_distance(a, b): return sum(abs(e1-e2) for e1, e2 in zip(a, b))

# calculate minkowski distance def minkowski\_distance(a, b, p): return sum(abs(e1-e2)"\*p for e1, e2 in zip(a, b)) \*\* (1/p)

# define data # actual values actual = [1, 0, 0, 1, 0, 0, 1, 0, 0, 1]

# predicted values predicted - [1, 0, 0, 1, 0, 0, 0, 1, 0, e]

# calculate distance

dist1 - euclidean\_distance(actual, predicted)

dist2 - manhattan\_distance(actual, predicted)

# calculate distance (p-1) st3 = minkowski distance(actual, predicted,1)

# calculate distance (p-1) dist3 = minkowski\_distance(actual, predicted,1)

print(dist3)

# calculate distance (p=2)

dist3 - minkowski\_distance(actual, predicted, 2) print(dist3)

# confusion matrix

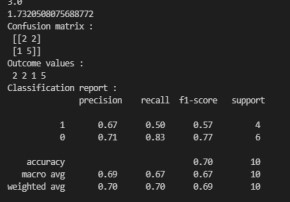
matrix = confusion\_matrix(actual, predicted, labels=[1, 0]) print('Confusion matrix : \n', matrix)

# outcome values order in sklearn tp, fn, fp, tn = confusion\_matrix(actual, predicted, labels=[1, 0]).reshape(-

1) print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy matrix - classification\_report(actual, predicted, labels-[1, 0]) print('Classification report : \n', matrix)

Output:



B. Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.

Code: import pandas as pd from sklearn.preprocessing import StandardScaler from sklearn.model\_selection import train\_test\_split from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix import warnings

warnings.filterwarnings('ignore')

# Xmatplotlib inline #Import the data set

raw\_data - pd.read\_csv('classified\_data.csv', index\_col - 0) print('07\_Govind\_saini')

print(raw\_data)

print(raw\_data.columns)

#Import standardization functions from scikit-learn

#Standardize the data set

scaler - StandardScaler() scaler.fit(raw\_data.drop('TARGET CLASS', axis-1))

scaled\_features - scaler.transform(raw\_data.drop('TARGET CLASS', axis-

1))

scaled\_data - pd.DataFrame(scaled\_features, columns - raw\_data.drop('TARGET CLASS', axis-1).columns)

scaler = Standardscaler()

scaler.fit(raw\_data.drop('TARGET CLASS', axis=1)) scaled\_features - scaler.transform(raw\_data.drop('TARGET CLASS', axis-

1))

scaled\_data = pd.DataFrame(scaled\_features, columns =

raw\_data.drop('TARGET CLASS', axis=1).columns)

#Split the data set into training data and test data

x = scaled\_data y = raw\_data['TARGET CLASS']

x\_training\_data, x\_test\_data, y\_training\_data, y\_test\_data - train\_test\_split(x, y, test\_size - 0.3)

#Train the model and make predictions model - KNeighborsClassifier(n\_neighbors - 1) model.fit(x\_training\_data, y\_training\_data)

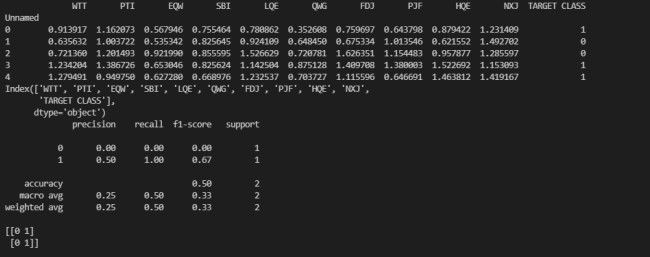
predictions = model.predict(x\_test\_data)

#Performance measurement

print(classification\_report(y\_test\_data, predictions))

print(confusion\_matrix(y\_test\_data, predictions))

Output:



# Practical-07

A. Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix

Code:

import matplotlib.pyplot as plt

import pandas as pd

#2 Importing the Mall\_Customers dataset by pandas dataset - pd.read\_csv('abalone.csv') X - dataset.iloc[:, [3,4]].values

import scipy.cluster.hierarchy as sch

print("07\_Govind Saini") #Lets create a dendrogram variable

dendrogram = sch.dendrogram(sch.linkage(X, method = "ward")) plt. title('Dendrogran') plt.xlabel('Gender') plt.ylabel('Euclidean distances') plt.show()

# algorithm class

from sklearn.cluster import AgglomerativeClustering hc - AgglomerativeClustering(n\_clusters - 5, affinity = 'euclidean', linkage ='ward')

# clusters vector that tells for each customer which cluster the customer belongs to. y\_hc=hc.fit\_predict(x)

print("Prediction Values :",y\_hc)

#5 Visualizing the clusters. This code is similar to k-means visualization code. plt.scatter(X[y\_hc -- 0, 0], x[y\_hc -- 0, 1], s-100, c-'red', label -'Cluster

1') plt.scatter(X[y\_hc -= 1, 0], x[y\_hc -- 1, 1], 5-100, C='blue', label

='Cluster 2')

pit. ticie( Denurogram ) plt.xlabel('Gender') plt.ylabel('Euclidean distances') plt.show()

# algorithm class

from sklearn.cluster import AgglomerativeClustering hc = AgglomerativeClustering(n\_clusters = 5, affinity = 'euclidean', linkage ='ward')

# clusters vector that tells for each customer which cluster the customer belongs to. y\_hc-hc.fit\_predict(x)

print("Prediction Values :",y\_hc)

#5 Visualizing the clusters. This code is similar to k-means visualization code.l plt.scatter(X[y\_hc -- 0, 0], x[y\_hc -- 0, 1], s-100, c-'red', label -'Cluster

1') plt.scatter(X[y\_hc -= 1, 0], x[y\_hc -- 1, 1], s-100, c='blue', label

='Cluster 2') plt.scatter(x[y\_hc == 2, 0], x[y\_hc == 2, 1], s=100, c='green', label

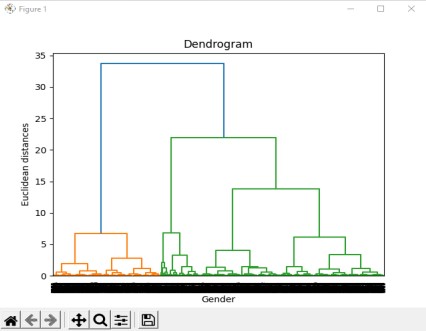
='Cluster 3') plt.scatter(x[y\_hc == 3, 0], x[y\_hc == 3, 1], s=100, C='cyan', label

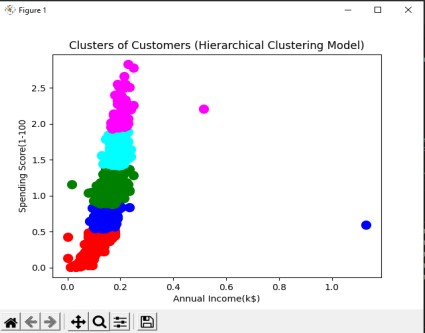
='Cluster 4') plt.scatter(X[y\_hc == 4, 0], x[y\_hc == 4, 1], s=100, C='magenta', label

='Cluster 5')

plt.title('Clusters of Customers (Hierarchical Clustering Model) ') plt.xlabel('Annual Income(k$)') plt.ylabel('Spending Score(1-100') plt.show()

Output:





# Practical-08

A. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Code: import numpy as np import pandas as pd

from pgmpy.estimators import MaximumLikelihoodEstimator from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

heartDisease = pd.read\_csv('heart\_disease\_data.csv') heartDisease - heartDisease.replace('?',np.nan)

print("07\_Govind Saini")

print('Sample instances from the dataset are given below') print(heartDisease.head())

print('\n Attributes and datatypes')

print(heartDisease.dtypes)

model- BayesianModel([('age', 'heartdisease'), ('gender',

'heartdisease'), ('exang', 'heartdisease'), ('cp', 'heartdisease'). print('\nLearning CPD using Maximum likelihood estimators') model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

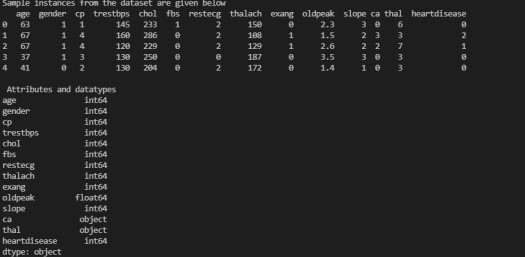
print('\n Inferencing with Bayesian Network:') HeartDiseasetest\_infer - VariableElimination(model)

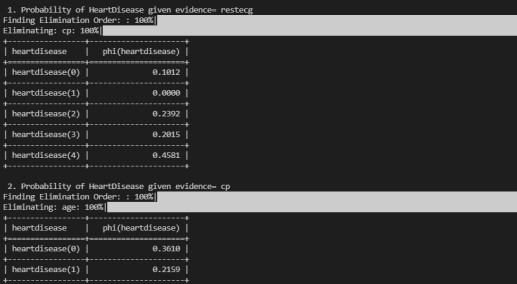
print('\n 1. Probability of HeartDisease given evidence= restecg') q1=HeartDiseasetest\_infer.query(variables=['heartdisease'],evidenc e={'restecg':1}) print(q1) print('\n Inferencing with Bayesian Network: ') HeartDiseasetest\_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence- restecg') q1-HeartDiseasetest\_infer.query(variables['heartdisease'],evidence-{'restecg':1]) print(q1)

print('\n 2. Probability of HeartDisease given evidence- cp ') q2-HeartDiseasetest\_infer.query(variables=['heartdisease'], evidence={'cp':2}) print(q2)

Output:





B. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Code:

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

def kernel(point, xmat, k): m,n = np.shape(xmat) weights = np.mat(np.eye((m))) for j in range(m): diff = point - x[j] weights[j,j] = np.exp(diff\*diff.T/(-2.0"k \*\* 2)) return weights

def localWeight(point, xmat, ymat, k): wei - kernel(point,xmat,k)

W = (X.T\*(wei\*X)).I\*(X.T\*(wei\*ymat.T))

return W

def localWeightRegression(xmat, ymat, k): m,n = np.shape(xmat) ypred = np. zeros(m) for i in range(m): ypred[1] = xmat[i]\*localWeight(xmat[i],xmat,ymat,k) return ypred

# load data points data = pd.read\_csv('user\_info.csv') bill - np.array(data.total\_user)

tip - np.array(data.tips)

#preparing and add 1 in bill mbill - np.mat(bill)

mtip = np.mat(tip)

m= np.shape(mbill)[1] one - np.mat(np.ones(m)) X - np.hstack((one.T,mbill.T))

#set k here

ypred = localWeightRegression(X,mtip,0.5) SortIndex - X[:,1].argsort(e) xsort = X[SortIndex][:,0] print("07\_Govind Saini") fig = plt.figure() ax - fig.add\_subplot(1,1,1) ax.scatter(bill, tip, color-'orange')

ax.plot(xsort[:,1],ypred[SortIndex], color = 'blue', linewidth-5) plt.xlabel('Total User') plt.ylabel('Tips')

plt.show()

Output:

