MACHINE LEARNING

GURU NANAK COLLEGE
OF ARTS, SCIENCE, AND COMMERCE
G.T.B NAGAR, SION, MUMBAI – 400037.

PRACTICAL OF

MACHINE LEARNING

SUBMITTED

BY

AQDAS AHMED KHAN

(M.Sc I.T Part II Sem III)

ACADEMIC YEAR 2023-2024

UNDER THE GUIDANCE OF

Mr. DHANRAJ JADHAV

Guru Nanak College of Arts, Science & Commerce
GTB Nagar, Mumbai – 37

Department of Information Technology

CERTIFICATE

This is to certify that **Mr. KHAN AQDAS AHMED**, Seat No. studying in **Master of Science in Information Technology Part II Semester III** has satisfactorily completed the Practical of **PSIT3P3a – MACHINE LEARNING** as prescribed by University of Mumbai, during the academic year **2023-24**.

__________________ __________________

Signature
Subject-In-Charge

Signature
Head of Department

____

Signature
External Examiner

College Seal: __________________ Date: __________________

INDEX

Sr. No.	Practical	Date	Signature
1	A. Design a simple machine learning model to train the training instances and test the same using Python. B. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data sample. Read the training data from a .CSV file.		
2	A. Perform Data Loading, Feature selection (Principal Component Analysis) and Feature Scoring and Ranking. B. For a given set of training data examples stored in .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.		
3	A. Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. B. Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.		
4	A. For a given set training data examples stored in a .CSV file implement Least Square Regression algorithm. (Use Univariate dataset) B. For a given set training data examples stored in a .CSV file implement Logistic Regression algorithm. (Use Multivariate dataset)		
5	A. Write a program to demonstrate the workings of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tress and apply this knowledge to classify a new sample. B. Write a program to implement K-Nearest Neighbour algorithm to classify the iris data set.		

6	A. Implement the different Distance methods (Euclidean, Manhattan Distance, Minkowski Distance) with Prediction, Test Score and Confusion Matrix. B. Implement the classification model using clustering for the following techniques with K mean clustering with Prediction, Test Score and Confusion Matrix.		
7	A. Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix. B. Implement the Rule based method and test the same.		
8	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. B. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experimental and draw graphs.		
9	A. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. B. Assuming a set of documents that need to be classified, use the Naïve Bayesian Classifier model to perform this task.		

Practical 1

Practical No. 1A:

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

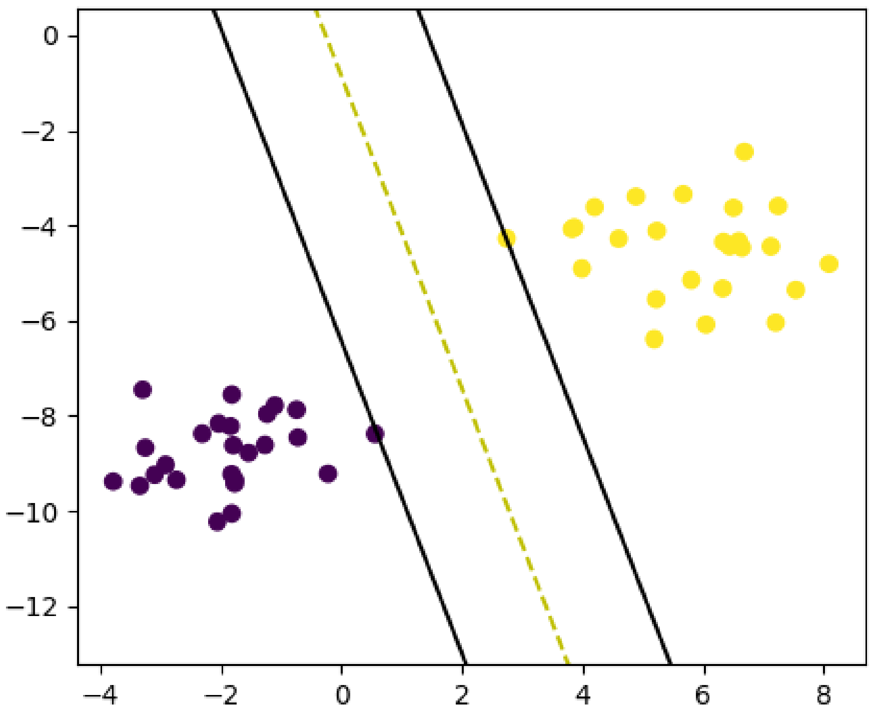
Code:

```
import numpy as np

print('Khan_Aqdas_Ahmed_03')

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)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w -
np.dot(x_i, y_[idx]))
                    self.b -= self.lr * y_[idx]

    def predict(self, X):
        approx = np.dot(X, self.w) - self.b
        return np.sign(approx)

if __name__ == "__main__":
    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)
    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")

        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:

```
al/Prac1A.py"
Khan_Aqdas_Ahmed_03
[0.58977016 0.17946483] -0.15200000000000001
```



Practical No. 1B:

Aim: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data sample. Read the training data from a .CSV file.

Datafile: 'data.csv'

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Honda	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Honda	Green	1980	Economy	Positive

Code:

```
import csv

num_attributes = 5
a = []
print('\n')
print('Khan_Aqdas_Ahmed_03')
print('\n The Given Training Data Set \n')

with open('data.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(0, 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:

```
Khan_Aqdas_Ahmed_03

 The Given Training Data Set

['Origin', 'Manufracturer', 'Color', 'Decade', 'Type', 'Example Type']
['Japan', 'Honda', 'Blue', '1980', 'Economy', 'Positive']
['Japan', 'Toyota', 'Green', '1970', 'Sports', 'Negative']
['Japan', 'Honda', 'Blue', '1990', 'Economy', 'Positive']
['USA', 'Chrysler', 'Red', '1980', 'Economy', 'Negative']
['Japan', 'Honda', 'White', '1980', 'Economy', 'Positive']
['Japan', 'Honda', 'Green', '1980', 'Economy', 'Positive']

 The initial value of hypothesis:
['0', '0', '0', '0', '0']

 The a[1] value of hypothesis:
['Japan', 'Honda', 'Blue', '1980', 'Economy']

 Find S: Finding a Maximally Specific Hypothesis

For Training instance No:0 the hypothesis is ['Japan', 'Honda', 'Blue', '1980', 'Economy']
For Training instance No:1 the hypothesis is ['Japan', 'Honda', 'Blue', '1980', 'Economy']
For Training instance No:2 the hypothesis is ['Japan', 'Honda', 'Blue', '1980', 'Economy']
For Training instance No:3 the hypothesis is ['Japan', 'Honda', 'Blue', '?', 'Economy']
For Training instance No:4 the hypothesis is ['Japan', 'Honda', 'Blue', '?', 'Economy']
For Training instance No:5 the hypothesis is ['Japan', 'Honda', '?', '?', 'Economy']
For Training instance No:6 the hypothesis is ['Japan', 'Honda', '?', '?', 'Economy']

 The Maximally Specific Hypothesis for a given Training Examples:
 ['Japan', 'Honda', '?', '?', 'Economy']
```

Practical 2

Practical No. 2A:

Aim: Perform Data Loading, Feature selection (Principal Component Analysis) and Feature Scoring and Ranking.

Datafile: [winequality-red.csv]()

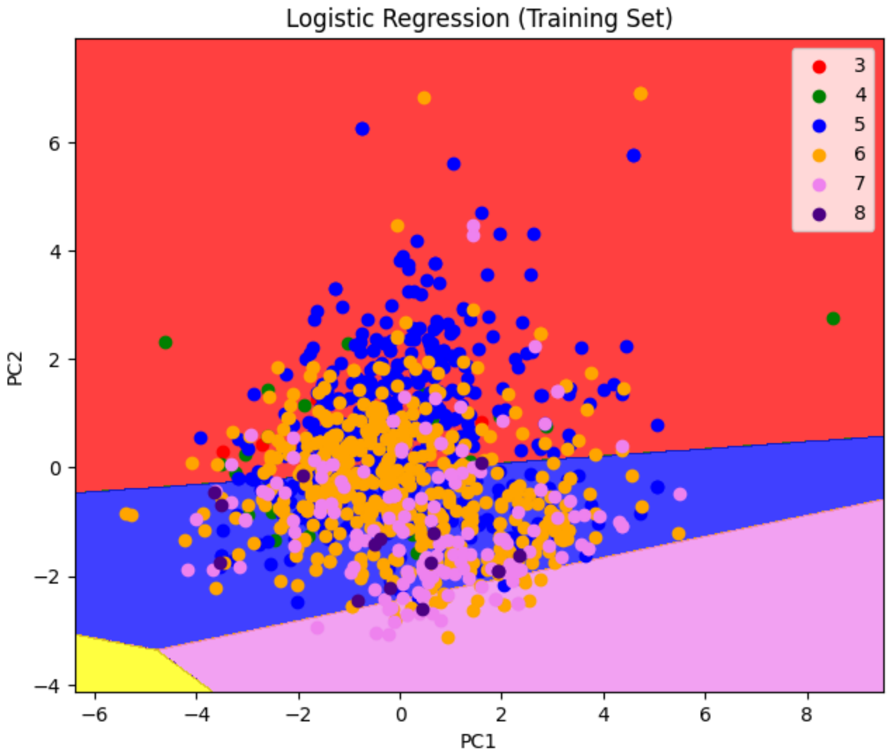
Code:

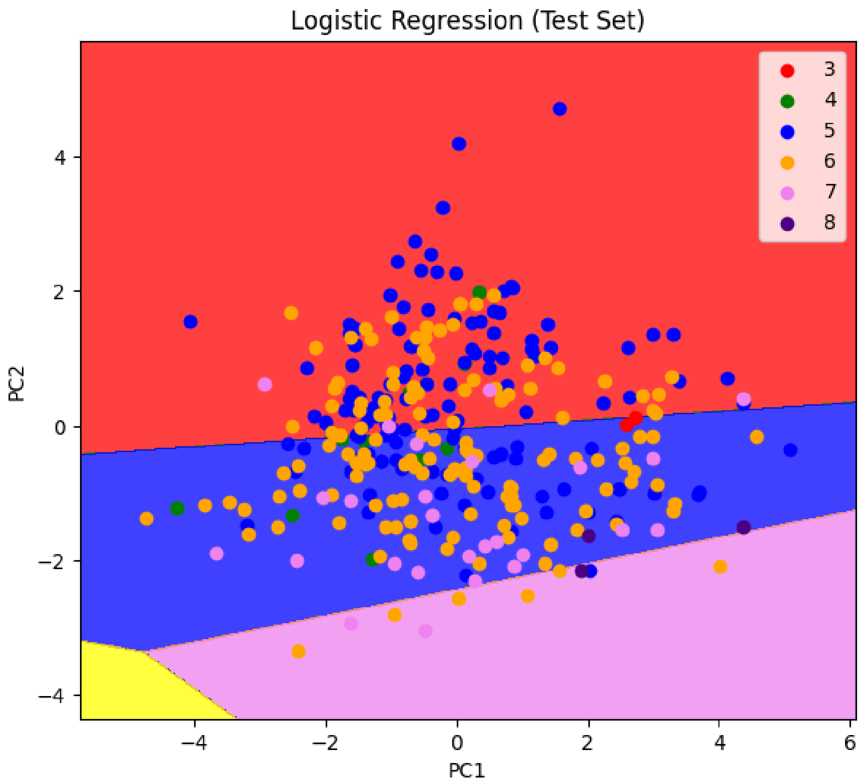
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from matplotlib.colors import ListedColormap

print("Khan_Aqdas_Ahmed_03")

dataset = pd.read_csv("winequality-red.csv")

X = dataset.iloc[:, 0:11].values
y = dataset.iloc[:, 11].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_

classifier = LogisticRegression(random_state=0, multi_class="multinomial")
classifier.fit(X_train_pca, y_train)
y_pred = classifier.predict(X_test_pca)

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

X_set, y_set = X_train_pca, y_train
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", "blue", "orange", "violet",
"indigo", "yellow")))

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],
        color=ListedColormap(("red", "green", "blue", "orange", "violet",
"indigo", "yellow"))(i), label=str(j))

plt.title("Logistic Regression (Training Set)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.show()

X_set, y_set = X_test_pca, 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", "blue", "orange", "violet",
"indigo", "yellow")))

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],
                color=ListedColormap(("red", "green", "blue", "orange",
"violet", "indigo", "yellow"))(i), label=str(j))

plt.title("Logistic Regression (Test Set)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.show()
```

Output:

```
Khan_Aqdas_Ahmed_03
Confusion Matrix:
[[ 0  0  0  2  0  0]
 [ 0  0  4  7  0  0]
 [ 0  0 89 45  1  0]
 [ 0  0 55 81  6  0]
 [ 0  0  4 21  2  0]
 [ 0  0  0  2  1  0]]
```





Practical No. 2B:

Aim: For a given set of training data examples stored in .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Datafile: ‘season_dataset.csv’

Sunny	Warm	Normal	Strong	Warm	Same
Sunny	Warm	High	Strong	Warm	Same
Sunny	Warm	High	Strong	Cool	Change

Code:

```
import csv
import numpy as np
import warnings
import matplotlib.pyplot as plt

print("Khan_Aqdas_Ahmed_03")

def g_0(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 != "0" and (x == y or y == "0"))
        more_general_parts.append(mg)
    return all(more_general_parts)

def fulfills(example, hypothesis):
    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] != '0' else x[i]
    return [tuple(h_new)]

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:]
```

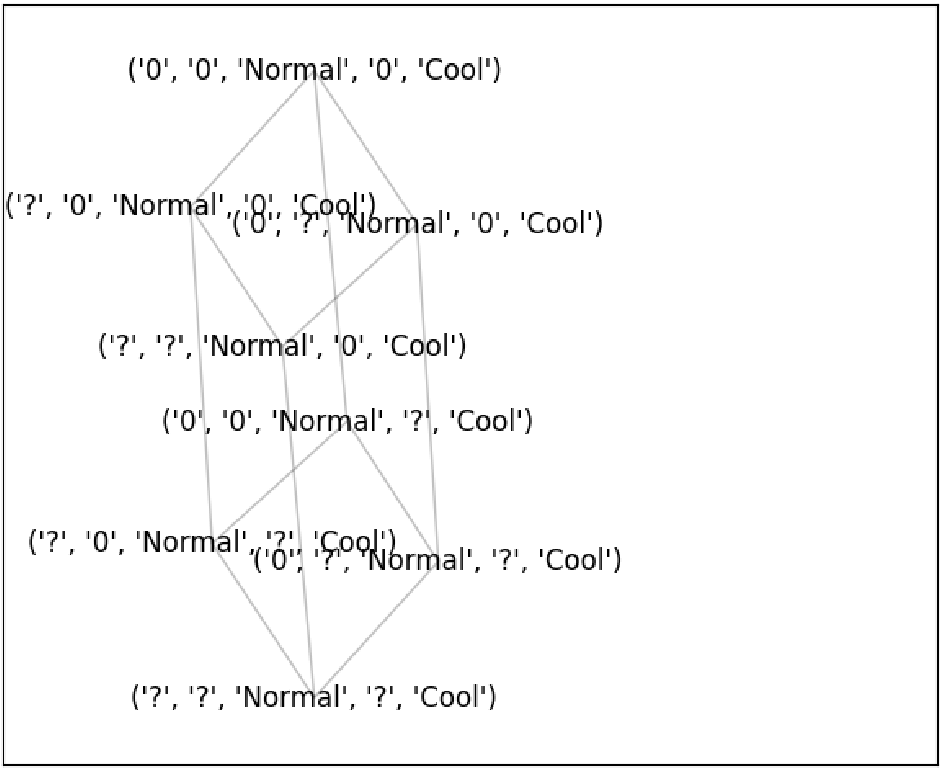
```
                    results.append(h_new)
            elif h[i] != "0":
                h_new = h[:i] + ('0',) + h[i+1:]
                results.append(h_new)
    return results

with open('season_dataset.csv') as csvFile:
    examples = [tuple(line) for line in csv.reader(csvFile)]

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]

def candidate_elimination(examples):
    domains = get_domains(examples)[:-1]

    G = set([g_0(len(domains))])
    S = set([s_0(len(domains))])

    for xcx in examples:
        x, cx = xcx[:-1], xcx[-1]
        if cx == 'Y':
            G = {g for g in G if fulfills(x, g)}
            S = generalize_S(x, G, S)
        else:
            S = {s for s in S if not fulfills(x, s)}
            G = specialize_G(x, domains, G, S)
    return G, 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)
            S.update([h for h in Splus if any([more_general(g, h) for g in
G])])

            S.difference_update([h for h in S if any([more_general(h, h1) for
h1 in S if h != h1])])
    return S

def specialize_G(x, domains, G, S):
    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)
            G.update([h for h in Gminus if any([more_general(h, s) for s in
S])])

            G.difference_update([h for h in G if any([more_general(g1, h) for
g1 in G if h != g1])])
    return G

G, S = candidate_elimination(examples)
print("G[4] = ", G)
print("S[4] = ", S)

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:]

    nextLvl = {}

    while True:
        for n in levels[-1]:
            for hyp in next_level(n.h, S):
                if hyp in nextLvl:
                    nextLvl[hyp].parents.add(n)
                else:
                    nextLvl[hyp] = HypothesisNode(hyp, curlevel, [n])
        if not nextLvl:
            break
        levels.append(list(nextLvl.values()))
        curlevel += 1
        nextLvl = {}
    return levels

def draw_hypothesis_space(G, S):
    import networkx as nx
```

```
    levels = build_hypothesis_space(G, S)

    g = nx.Graph()

    for nodes in levels:
        for n in nodes:
            for p in n.parents:
                g.add_edge(n.h, p.h)

    pos = {}
    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, S)
```

Output:

```
Khan_Aqdas_Ahmed_03
G[4] =  {('?', '?', 'Normal', '?', 'Cool')}
S[4] =  {('0', '0', '0', '0', '0')}
```


('0', '0', 'Normal', '0', 'Cool')
('?', '0', 'Normal', '0', 'Cool')
('0', '?', 'Normal', '0', 'Cool')
('?', '?', 'Normal', '0', 'Cool')
('0', '0', 'Normal', '?', 'Cool')
('?', '0', 'Normal', '?', 'Cool')
('0', '?', 'Normal', '?', 'Cool')
('?', '?', 'Normal', '?', 'Cool')

Practical 3

Practical No. 3A:

Aim: Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Datafile: tennisdata.csv

Code:

```
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

print("Khan_Aqdas_Ahmed_03")

data = pd.read_csv('tennisdata.csv')
print("The first 5 Values of data is:\n ", data.head())

X = data.iloc[:,:-1]
print("\nThe first 5 values of train data is: \n", X.head())

y = data.iloc[:, -1]
print("\nThe first 5 values of train output is: \n", y.head())

le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)

print("\nNow the train data is: \n", X.head())

le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the train outpur is: \n", y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

classifier = GaussianNB()
```

```
classifier.fit(X_train, y_train)

print("Accuracy is: ", accuracy_score(classifier.predict(X_test), y_test))
```

Output:

```
Khan_Aqdas_Ahmed_03
The first 5 Values of data is:
    Outlook Temperature Humidity  Windy PlayTennis
0     Sunny         Hot     High  False         No
1     Sunny         Hot     High   True         No
2  Overcast         Hot     High  False        Yes
3     Rainy        Mild     High  False        Yes
4     Rainy        Cool   Normal  False        Yes

The first 5 values of train data is:
    Outlook Temperature Humidity  Windy
0     Sunny         Hot     High  False
1     Sunny         Hot     High   True
2  Overcast         Hot     High  False
3     Rainy        Mild     High  False
4     Rainy        Cool   Normal  False

The first 5 values of train output is:
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object

Now the train data is:
   Outlook  Temperature  Humidity  Windy
0        2            1         0      0
1        2            1         0      1
2        0            1         0      0
3        1            2         0      0
4        1            0         1      0

Now the train output is:
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is:  0.6666666666666666
```

Practical No. 3B:

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

Datafile: [HR-Employee-Attrition-All.csv]()

Code:

```
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.tree import DecisionTreeClassifier

print("Khan_Aqdas_Ahmed_03")

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.append(column)

df['Attrition'] = df.Attrition.astype("category").cat.codes

label = LabelEncoder()
for column in categorical_col:
    df[column] = label.fit_transform(df[column])

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)

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("Train Result: \n===================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
```

```python
        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")

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:

```
T/MScIT Part-2/ML/Practical/Prac3B.py
Khan_Aqdas_Ahmed_03
Train Result:
====================
Accuracy Score: 100.00%
CLASSIFICATION REPORT:
                0      1  accuracy  macro avg  weighted avg
precision     1.0    1.0       1.0        1.0           1.0
recall        1.0    1.0       1.0        1.0           1.0
f1-score      1.0    1.0       1.0        1.0           1.0
support     853.0  176.0       1.0     1029.0        1029.0
Confusion Matrix:
[[853   0]
 [  0 176]]

Test Result:
====================
Accuracy Score: 77.78%
CLASSIFICATION REPORT:
                     0          1  accuracy   macro avg  weighted avg
precision     0.887363   0.259740  0.777778    0.573551      0.800549
recall        0.850000   0.327869  0.777778    0.588934      0.777778
f1-score      0.868280   0.289855  0.777778    0.579067      0.788271
support     380.000000  61.000000  0.777778  441.000000    441.000000
Confusion Matrix:
[[323  57]
 [ 41  20]]
```

Practical 4

Practical No. 4A:

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

Datafile: 'Sample_Salary_Data.csv'

YearsOfExperience	SalaryIn1000s
2	15
3	28
5	42
15	64
8	50
16	90
11	58
1	8
9	54

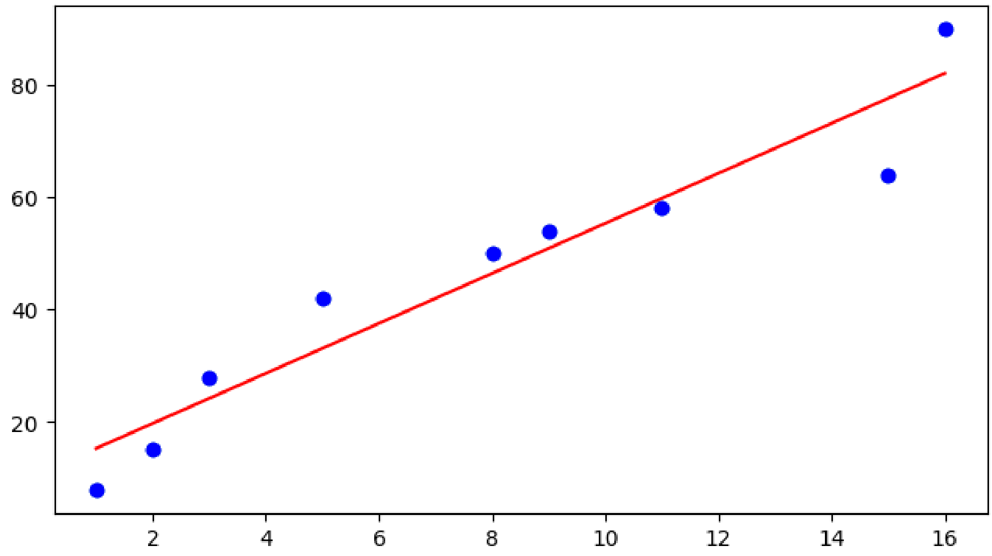
Code:

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

print("Khan_Aqdas_Ahmed_03")

plt.rcParams['figure.figsize'] = (12.0, 9.0)

data = pd.read_csv('Sample_Salary_Data.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
plt.scatter(X, Y)

X_mean = np.mean(X)
Y_mean = np.mean(Y)
num = 0
den = 0

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(m, c)

Y_pred = m * X + c

plt.scatter(X, Y, color = 'blue')
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color = 'red')
```

```
plt.show()
```

Output:

```
Khan_Aqdas_Ahmed_03
4.44986200551978 10.834406623735049
```



Practical No. 4B:

Aim: For a given set training data examples stored in a .CSV file implement Logistic Regression algorithm. (Use Multivariate dataset)

Datafile: ‘p4b.csv’

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1

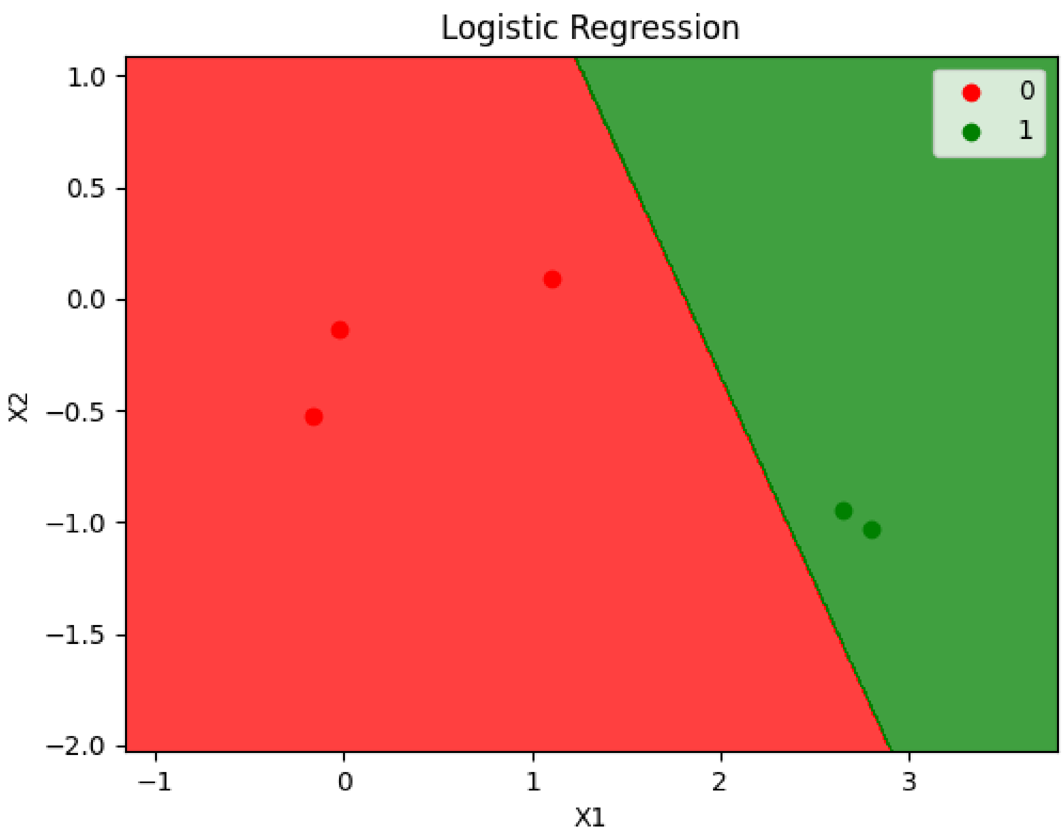
Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
from matplotlib.colors import ListedColormap
import warnings
warnings.filterwarnings('ignore')

print("Khan_Aqdas_Ahmed_03")

dataset = pd.read_csv('p4b.csv')
X = dataset.iloc[:,[2, 3]].values
y = dataset.iloc[:, 4].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train[0:10, :])

classifier = LogisticRegression()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix: \n", cm)

print("Accuracy: ", accuracy_score(y_test, y_pred))

df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})

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_test[y_test == j, 0], X_test[y_test == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Logistic Regression')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.show()
```

Output:

```
Khan_Aqdas_Ahmed_03
[[-0.16096873  0.51211883]
 [ 0.68411709  2.4725737 ]
 [-1.00605455  0.68015781]
 [ 1.10666    -1.16827107]
 [-1.14690218 -1.19627757]
 [-0.16096873 -0.27206313]
 [-0.02012109 -0.10402414]
 [ 0.68411709 -1.22428407]
 [-1.28774982  0.56813182]
 [ 2.51513636 -1.00023208]]
Confusion Matrix:
 [[3 0]
 [0 2]]
Accuracy:  1.0
```



Practical 5

Practical No. 5A:

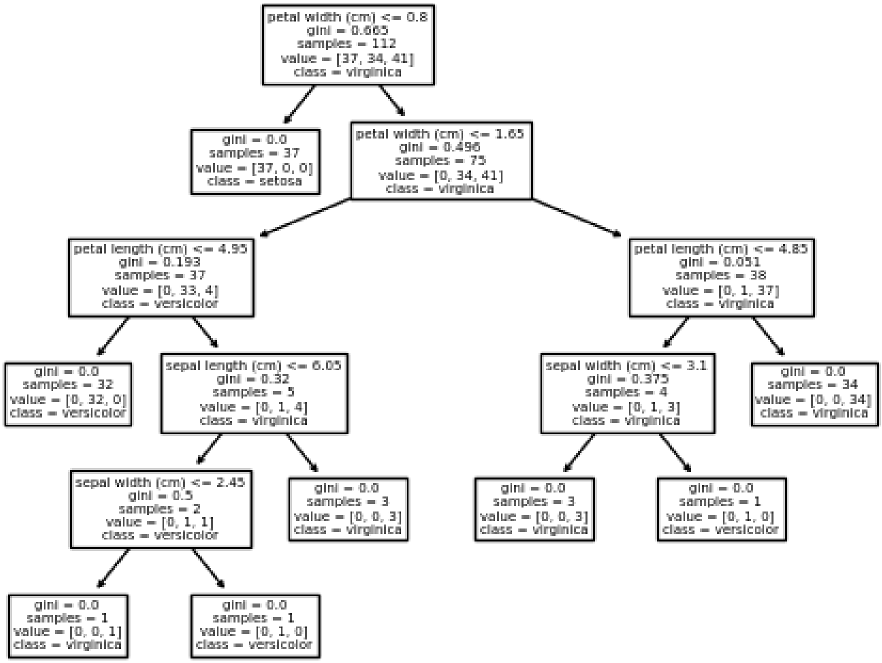
Aim: Write a program to demonstrate the workings of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision trees and apply this knowledge to classify a new sample.

Code:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier, export_graphviz, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt

print("Khan_Aqdas_Ahmed_03")

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print("Confusion Matrix: \n", confusion_matrix(y_test, y_pred))
print("Accuracy: ", accuracy_score(y_test, y_pred))

plot_tree(clf, feature_names=iris.feature_names,
class_names=iris.target_names)
plt.show()
```

Output:

```
Khan_Aqdas_Ahmed_03
Confusion Matrix:
 [[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
Accuracy:  0.9736842105263158
```



Practical No. 5B:

Aim: 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, GridSearchCV

print("Khan_Aqdas_Ahmed_03")

iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

knn = KNeighborsClassifier(n_neighbors=5, p=2, weights='uniform',
algorithm='auto')
knn.fit(X_train_std, y_train)
```

```python
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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)

pipeline = make_pipeline(StandardScaler(), KNeighborsClassifier())

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'],
}]

gs = GridSearchCV(pipeline, param_grid=param_grid,
                  scoring='accuracy',
                  refit=True,
                  cv=10,
                  verbose=1,
                  n_jobs=2)

gs.fit(X_train, y_train)

print('Best Score: %.3f' % gs.best_score_,
      '\nBest Parameters: ', gs.best_params_)
```

Output:

```
Part-2/ML/Practical/Prac5B.py
Khan_Aqdas_Ahmed_03
Training accuracy score: 0.981
Test accuracy score: 0.911
Fitting 10 folds for each of 144 candidates, totalling 1440 fits
Best Score: 0.972
Best Parameters:  {'kneighborsclassifier__algorithm': 'auto', 'kn
eighborsclassifier__n_neighbors': 5, 'kneighborsclassifier__p': 1
, 'kneighborsclassifier__weights': 'uniform'}
```

Practical 6

Practical No. 6A:

Aim: 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, classification_report

print("Khan_Aqdas_Ahmed_03")

def euclidean_distance(a, b):
    return sqrt(sum((e1-e2) ** 2 for e1, e2 in zip(a, b)))

def manhattan_distance(a, b):
    return sum(abs(e1 - e2) for e1, e2 in zip(a, b))

def minkowski_distance(a, b, p):
    return sum(abs(e1 - e2) ** p for e1, e2 in zip(a, b)) ** (1/p)

actual = [1, 0, 0, 1, 0, 0, 1, 0, 0, 1]

predicted = [1, 0, 0, 1, 0, 0, 0, 1, 0, 0]

dist1 = euclidean_distance(actual, predicted)
print("Euclidean Distance: ", dist1)

dist2 = manhattan_distance(actual, predicted)
print("Manhattan Distance: ", dist2)

dist3 = minkowski_distance(actual, predicted, 1)
print("Minkowski Distance wiht p = 1: ", dist3)

dist3 = minkowski_distance(actual, predicted, 2)
print("Minkowski Distance with p = 2: ", dist3)

matrix = confusion_matrix(actual, predicted, labels=[1, 0])
print("Confusion Matrix: \n", matrix)

tp, fn, fp, tn = confusion_matrix(actual, predicted, labels=[1, 0]).reshape(-
1)
print("Outcome values: \n", tp, fn, fp, tn)

matrix = classification_report(actual, predicted, labels=[1, 0])
print("Classificaiton report: \n", matrix)
```

Output:

```
T/MSCIT Part-2/ML/Practical/Prac6A.py
Khan_Aqdas_Ahmed_03
Euclidean Distance:  1.7320508075688772
Manhattan Distance:  3
Minkowski Distance wiht p = 1:  3.0
Minkowski Distance with p = 2:  1.7320508075688772
Confusion Matrix:
 [[2 2]
 [1 5]]
Outcome values:
 2 2 1 5
Classificaiton report:
               precision    recall  f1-score   support

           1       0.67      0.50      0.57         4
           0       0.71      0.83      0.77         6

    accuracy                           0.70        10
   macro avg       0.69      0.67      0.67        10
weighted avg       0.70      0.70      0.69        10
```

Practical No. 6B:

Aim: Implement the classification model using clustering for the following techniques with K mean clustering with Prediction, Test Score and Confusion Matrix.

Datafile: classified_data.csv

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, confusion_matrix
import warnings
warnings.filterwarnings('ignore')

print("Khan_Aqdas_Ahmed_03")

raw_data = pd.read_csv('Classified Data.csv', index_col=0)
print(raw_data.head())
print(raw_data.columns)

scaler = StandardScaler()
scaler.fit(raw_data.drop('TARGET CLASS', axis=1))
scaled_features = scaler.transform(raw_data.drop('TARGET CLASS', axis=1))
```

```python
scaled_data = pd.DataFrame(scaled_features, columns=raw_data.drop('TARGET CLASS', axis=1).columns)

x = scaled_data
y = raw_data['TARGET CLASS']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

model = KNeighborsClassifier(n_neighbors=1)
model.fit(x_train, y_train)
predictions = model.predict(x_test)

print(classification_report(y_test, predictions))
print(confusion_matrix(y_test, predictions))
```

Output:

```
Khan_Aqdas_Ahmed_03
        WTT       PTI       EQW       SBI  ...       PJF       HQE       NXJ  TARGET CLASS
0  0.913917  1.162073  0.567946  0.755464  ...  0.643798  0.879422  1.231409             1
1  0.635632  1.003722  0.535342  0.825645  ...  1.013546  0.621552  1.492702             0
2  0.721360  1.201493  0.921990  0.855595  ...  1.154483  0.957877  1.285597             0
3  1.234204  1.386726  0.653046  0.825624  ...  1.380003  1.522692  1.153093             1
4  1.279491  0.949750  0.627280  0.668976  ...  0.646691  1.463812  1.419167             1

[5 rows x 11 columns]
Index(['WTT', 'PTI', 'EQW', 'SBI', 'LQE', 'QWG', 'FDJ', 'PJF', 'HQE', 'NXJ',
       'TARGET CLASS'],
      dtype='object')
              precision    recall  f1-score   support

           0       0.97      0.87      0.92       163
           1       0.86      0.97      0.91       137

    accuracy                           0.91       300
   macro avg       0.92      0.92      0.91       300
weighted avg       0.92      0.91      0.91       300

[[141  22]
 [  4 133]]
```

Practical 7

Practical No. 7A:

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

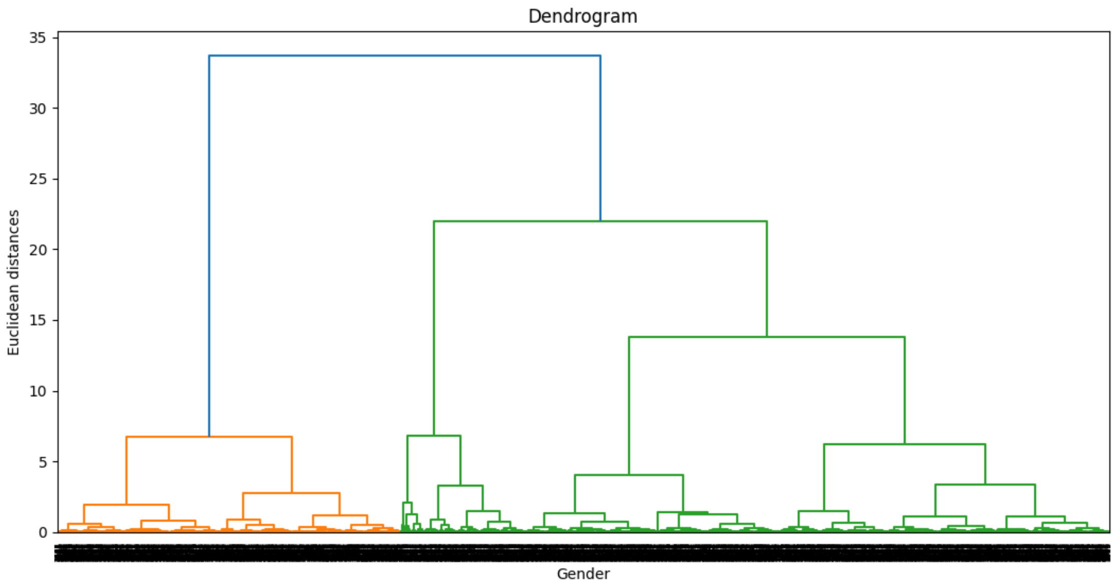
Datafile: [abalone.csv]

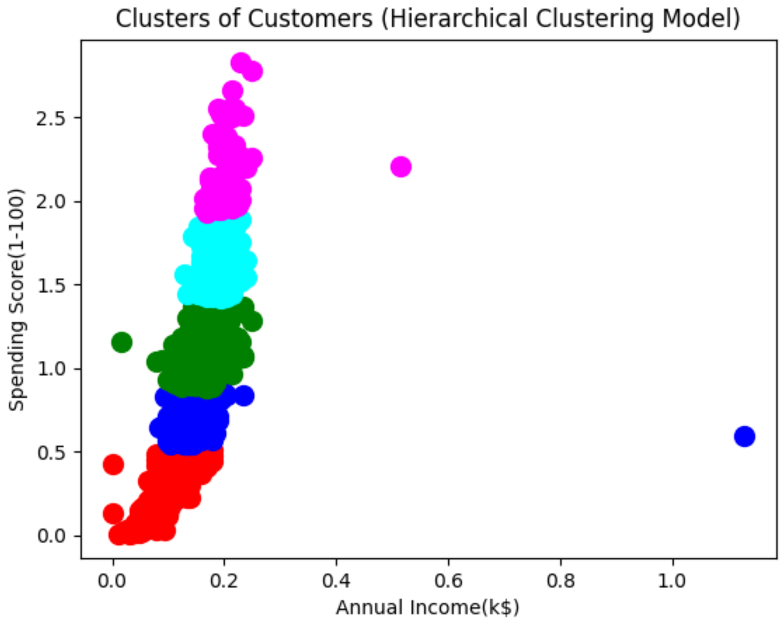
Code:

```
import matplotlib.pyplot as plt
import pandas as pd
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering

print("Khan_Aqdas_Ahmed_03")

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

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


hc = AgglomerativeClustering(n_clusters=5, affinity='euclidean',
linkage='ward')

y_hc = hc.fit_predict(X)
print("Prediction Values: ", y_hc)

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:

```
Khan_Aqdas_Ahmed_03
Prediction Values:  [0 1 0 ... 2 2 4]
```





Practical No. 7B:

Aim: Implement the Rule based method and test the same.

Code:

```
import pandas as pd
from sklearn.metrics import accuracy_score, confusion_matrix

print("Khan_Aqdas_Ahmed_03")

# Sample data
data = {
    'Tid': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Refund': ['yes', 'no', 'no', 'yes', 'no', 'no', 'yes', 'no', 'no', 'no'],
    'Marital_Status': ['single', 'married', 'single', 'married', 'divorced',
'married', 'divorced', 'married', 'single', 'married'],
```

```
    'Taxable_Income': [125, 100, 70, 120, 65, 70, 200, 85, 75, 90],
    'Class': ['no', 'no', 'no', 'no', 'yes', 'no', 'no', 'yes', 'no', 'yes']
}

df = pd.DataFrame(data)

# Rule-based method
def rule_based_classifier(row):
    if row['Refund'] == 'yes' and row['Marital_Status'] == 'single' and
row['Taxable_Income'] > 100:
        return 'yes'
    elif row['Refund'] == 'no' and row['Marital_Status'] == 'married' and
row['Taxable_Income'] <= 80:
        return 'yes'
    else:
        return 'no'

# Apply the rule-based classifier to the DataFrame
df['Prediction'] = df.apply(rule_based_classifier, axis=1)

# Evaluate the model
accuracy = accuracy_score(df['Class'], df['Prediction'])
conf_matrix = confusion_matrix(df['Class'], df['Prediction'])

print("Khan_Aqdas_Ahmed")

# Print results
print(f'Accuracy: {accuracy}')
print('Confusion Matrix:')
print(conf_matrix)
print('\nPrediction Values:')
print(df[['Tid', 'Class', 'Prediction']])
```

Output:

```
Khan_Aqdas_Ahmed
Accuracy: 0.5
Confusion Matrix:
[[5 2]
 [3 0]]

Prediction Values:
   Tid Class Prediction
0    1    no        yes
1    2    no         no
2    3    no         no
3    4    no         no
4    5   yes         no
5    6    no        yes
6    7    no         no
7    8   yes         no
8    9    no         no
9   10   yes         no
PS C:\Users\Anas\Desktop\Aqdas
```

Practical 8

Practical No. 8A:

Aim: 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.

Datafile: [heart_disease_data.csv]

Code:

```
import numpy as np
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianNetwork
from pgmpy.inference import VariableElimination
import warnings
warnings.filterwarnings('ignore')

print("Khan_Aqdas_Ahmed_03")

heartDisease = pd.read_csv('heart_disease_data.csv')
heartDisease = heartDisease.replace('?', np.nan)

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

print('\nAttributes and datatypes')
print(heartDisease.dtypes)

model = BayesianNetwork([('age', 'target'), ('sex', 'target'), ('exang',
'target'), ('cp', 'target'), ('restecg', 'target'), ('trestbps', 'target'),
('chol', 'target'), ('fbs', 'target')])
print("\nLearning CPD using maximum likelihood estimators")
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

print("\nInferencing with Bayesian Network: ")
HeartDiseasetest_infer = VariableElimination(model)

print("\n 1. Probability of Heart Disease given evidance = restecg")
q1 = HeartDiseasetest_infer.query(variables=['target'],
evidence={'restecg':1})
print(q1)

print("\n 2. Probability of Heart Disease given evidance = cp")
q2 = HeartDiseasetest_infer.query(variables=['target'], evidence={'cp':2})
print(q2)
```

Output:

```
Khan_Aqdas_Ahmed_03
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  ...  exang  oldpeak  slope  ca  thal  target
0   63    1   3       145   233    1  ...      0      2.3      0   0     1       1
1   37    1   2       130   250    0  ...      0      3.5      0   0     2       1
2   41    0   1       130   204    0  ...      0      1.4      2   0     2       1
3   56    1   1       120   236    0  ...      0      0.8      2   0     2       1
4   57    0   0       120   354    0  ...      1      0.6      2   0     2       1

[5 rows x 14 columns]

Attributes and datatypes
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object

Learning CPD using maximum likelihood estimators

Inferencing with Bayesian Network:

 1. Probability of Heart Disease given evidance = restecg
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.5000 |
+-----------+---------------+
| target(1) |        0.5000 |
+-----------+---------------+

 2. Probability of Heart Disease given evidance = cp
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.5000 |
+-----------+---------------+
| target(1) |        0.5000 |
+-----------+---------------+
```

Practical No. 8B:

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

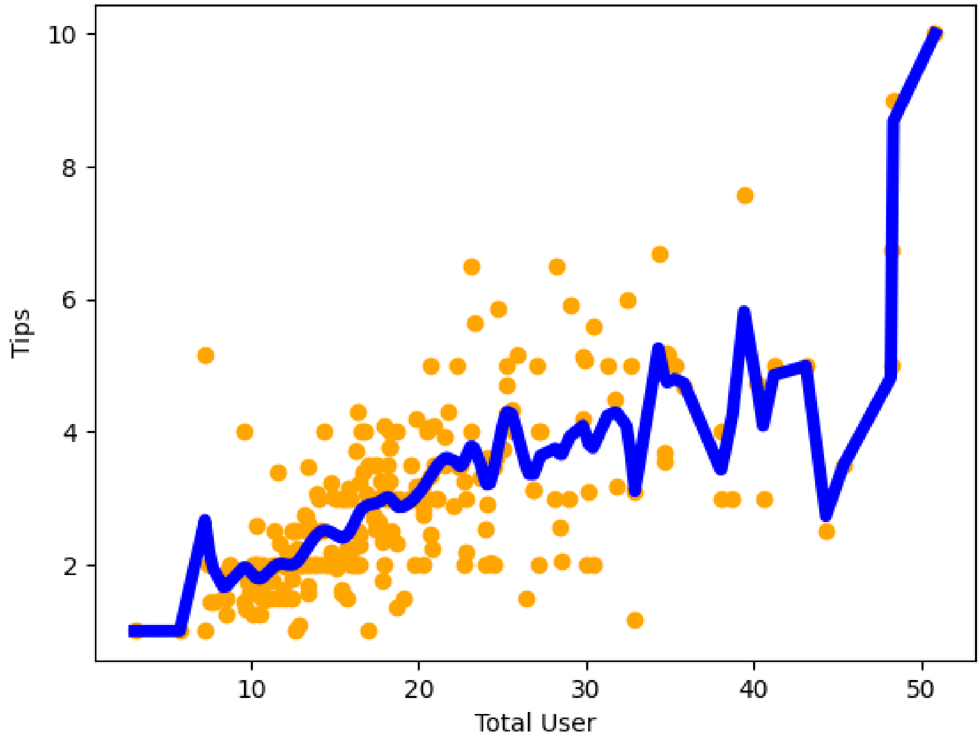
Datafile: [10-dataset.csv]()

Code:

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

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

print("Khan_Aqdas_Ahmed_03")

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[i] = xmat[i]*localWeight(xmat[i], xmat, ymat, k)
    return ypred

data = pd.read_csv('10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

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))

ypred = localWeightRegression(X, mtip, 0.5)
SortIndex = X[:, 1].argsort(0)
xsort = X[SortIndex][:, 0]

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:


10
8
6
4
2
Tips
10
20
30
40
50
Total User

Practical 9

Practical No. 9A:

Aim: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Code:

```
import random
from math import exp
from random import seed

print("Khan_Aqdas_Ahmed_03")

def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights': [random.uniform(-0.5, 0.5) for i in
range(n_inputs + 1)]} for i in range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{'weights': [random.uniform(-0.5, 0.5) for i in
range(n_hidden + 1)]} for i in range(n_outputs)]
    network.append(output_layer)
    i = 1
    print("\n The initialized Neural Network:\n")
    for layer in network:
        j = 1
        for sub in layer:
            print("\n Layer[%d] Node[%d]:\n" %(i,j), sub)
            j = j+1
        i = i+1
    return network

def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
             activation = activate(neuron['weights'], inputs)
             neuron['output'] = transfer(activation)
             new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs
```

```
def transfer_derivative(output):
    return output * (1.0 - output)

def backward_propaget_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()

        if i != len(network) - 1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])

        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] *
transfer_derivative(neuron['output'])

def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0 :
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']

#Train
def train_network(network, train, l_rate, n_epoch, n_outputs):
    print("\n Network Training Begins:\n")

    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i] - outputs[i]) ** 2 for i in
range(len(expected))])
            backward_propaget_error(network, expected)
            update_weights(network, row, l_rate)
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate,
sum_error))
    print("\n Network Training Ends: ")
```

```python
#Test
seed(2)
dataset = [[2.7810836,2.550537003,0],
           [1.465489372,2.362125076,0],
           [3.396561688,4.400293529,0],
           [1.38807019,1.850220317,0],
           [3.06407232,3.005305973,0],
           [7.627531214,2.759262235,1],
           [5.332441248,2.088626775,1],
           [6.922596716,1.77106367,1],
           [8.675418651,-0.242068655,1],
           [7.673756466,3.508563011,1]]

print("\n The input Data Set: \n", dataset)
n_inputs = len(dataset[0]) - 1
print("\n Number of Inputs: \n", n_inputs)
n_outputs = len(set([row[-1] for row in dataset]))
print("\n Number of Outputs: \n", n_outputs)

network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)

print("\n Final Neural Network: ")
i = 1
for layer in network:
    j = 1
    for sub in layer:
        print("\n Layer[%d] Node[%d]: \n" %(i, j), sub)
        j = j + 1
    i = i + 1
```

Output:

```
Khan_Aqdas_Ahmed_03

 The input Data Set: 
 [[2.7810836, 2.550537003, 0], [1.465489372, 2.362125076, 0], [3.396561688, 4.400293529, 0], [
1.38807019, 1.850220317, 0], [3.06407232, 3.005305973, 0], [7.627531214, 2.759262235, 1], [5.3
32441248, 2.088626775, 1], [6.922596716, 1.77106367, 1], [8.675418651, -0.242068655, 1], [7.67
3756466, 3.508563011, 1]]

 Number of Inputs:  2

 Number of Outputs:  2

 The initialized Neural Network:

 Layer[1] Node[1]:
 {'weights': [0.4560342718892494, 0.4478274870593494, -0.4434486322731913]}

 Layer[1] Node[2]:
 {'weights': [-0.41512800484107837, 0.33549887812944956, 0.2359699890685233]}

 Layer[2] Node[1]:
 {'weights': [0.1697304014402209, -0.1918635424108558, 0.10594416567846243]}

 Layer[2] Node[2]:
 {'weights': [0.10680173364083789, 0.08120401711200309, -0.3416171297451944]}
```

```
Network Training Begins:

>epoch=0, lrate=0.500, error=5.278
>epoch=1, lrate=0.500, error=5.122
>epoch=2, lrate=0.500, error=5.006
>epoch=3, lrate=0.500, error=4.875
>epoch=4, lrate=0.500, error=4.700
>epoch=5, lrate=0.500, error=4.466
>epoch=6, lrate=0.500, error=4.176
>epoch=7, lrate=0.500, error=3.838
>epoch=8, lrate=0.500, error=3.469
>epoch=9, lrate=0.500, error=3.089
>epoch=10, lrate=0.500, error=2.716
>epoch=11, lrate=0.500, error=2.367
>epoch=12, lrate=0.500, error=2.054
>epoch=13, lrate=0.500, error=1.780
>epoch=14, lrate=0.500, error=1.546
>epoch=15, lrate=0.500, error=1.349
>epoch=16, lrate=0.500, error=1.184
>epoch=17, lrate=0.500, error=1.045
>epoch=18, lrate=0.500, error=0.929
>epoch=19, lrate=0.500, error=0.831

Network Training Ends:

Final Neural Network:

Layer[1] Node[1]:
{'weights': [0.8642508164347664, -0.8497601716670761, -0.8668929014392035], 'output': 0.92955
87965836384, 'delta': 0.005645382825629247}

Layer[1] Node[2]:
{'weights': [-1.2934302410111027, 1.7109363237151511, 0.7125327507327331], 'output': 0.047607
03296164143, 'delta': -0.005928559978815065}

Layer[2] Node[1]:
{'weights': [-1.3098359335096292, 2.16462207144596, -0.3079052288835877], 'output': 0.1989556
395205846, 'delta': -0.03170801648036036}

Layer[2] Node[2]:
{'weights': [1.5506793402414165, -2.11315950446121, 0.1333585709422027], 'output': 0.80950426
53312078, 'delta': 0.029375796661413225}
```

Practical No. 9B:

Aim: Assuming a set of documents that need to be classified, use the Naïve Bayesian Classifier model to perform this task.

Datafile: document.csv

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,
recall_score

print("Khan_Aqdas_Ahmed_03")
```

```python
msg = pd.read_csv('document.csv', names=['message', 'label'])
print("Total Instances of Dataset: ", msg.shape[0])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})

x = msg.message
y = msg.labelnum

xtrain, xtest, ytrain, ytest = train_test_split(x, y)

count_v = CountVectorizer()
xtrain_dm = count_v.fit_transform(xtrain)
xtest_dm = count_v.transform(xtest)

df = pd.DataFrame(xtrain_dm.toarray(), columns =
count_v.get_feature_names_out())
print(df[0:5])

clf = MultinomialNB()
clf.fit(xtrain_dm, ytrain)
pred = clf.predict(xtest_dm)

for doc, p in zip(xtrain, pred):
    p = 'pos' if p == 1 else 'neg'
    print("%s -> %s" % (doc, p))

print("Accuracy Matrics: \n")
print("Accuracy: ", accuracy_score(ytest, pred))
print("Recall: ", recall_score(ytest, pred))
print("Precision: ", precision_score(ytest, pred))
print("Confusion Matrix: \n", confusion_matrix(ytest, pred))
```

Output:

```
T/MScIT Part-2/ML/Practical/Prac9B.py
Khan_Aqdas_Ahmed_03
Total Instances of Dataset:  18
   about  am  an  awesome  beers  can  dance  ...  very  view  we  went  what  will  with
0      0   1   0        0      0    0      0  ...     0     0   0     0     0     0     0
1      0   0   0        0      0    0      0  ...     0     0   0     0     0     0     0
2      0   0   1        1      0    0      0  ...     0     0   0     0     0     0     0
3      1   0   0        0      1    0      0  ...     1     0   0     0     0     0     0
4      0   0   0        0      0    0      0  ...     0     0   0     0     0     0     0

[5 rows x 45 columns]
I am tired of this stuff -> neg
I do not like this restaurant -> neg
This is an awesome place -> pos
I feel very good about these beers -> neg
I love this sandwich -> pos
Accuracy Matrics:

Accuracy:  0.6
Recall:  0.5
Precision:  0.5
Confusion Matrix:
 [[2 1]
 [1 1]]
```