Dataset

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Output

[['Sunny', 'Rainy'], ['Warm', 'Cold'], ['Normal', 'High'], ['Strong', 'Weak'], ['Warm', 'Cool'], ['Same', 'Change']]

6

Date

Expt. No. 01

Page No. 02

LAB PROGRAM-1

1) Implement and demonstrate 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

```
import random
import csv
attributes = [['Sunny', 'Rainy'],
              ['Warm', 'Cold'],
              ['Normal', 'High'],
              ['Strong', 'Weak'],
              ['Warm', 'Cool'],
              ['Same', 'Change']]
print(attributes)
num_attributes = len(attributes)
print(num_attributes)
print("\n The most general hypothesis: ['?', '?', '?', '?', '?', '?']\n")
print("\n The most specific hypothesis: ['0', '0', '0', '0', '0', '0']\n")
a=[]
print("\n The given training data set \n")
with open('sport.csv','r') as csvfile:
    reader= csv.reader(csvfile)
    for row in reader:
        a.append(row)
        print (row)
print("\n The initial value of hypothesis")
h=['0'] * num_attributes
print(h)
for i in range(0,len(a)):
```

Teacher's Signature

The most general hypothesis :['?','?','?','?','?','?']

The most specific hypothesis:['0','0','0','0','0','0']

The given training data set

['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast', 'EnjoySport']

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']

['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']

['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']

['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

The initial value of hypothesis

['0', '0', '0', '0', '0', '0']

for training example: 5 the hypothesis

['Sunny', 'Warm', '?', 'Strong', '?', '?']

Date

Expt. No. 03

Page No. 04

```
if a[i][num_attributes]=='Yes':
    for j in range(num_attributes):
        if h[j]=='0' or h[j]==a[i][j]:
            h[j]=a[i][j]
        else:
            h[j]='?'
print("\n for training example:{0} the hypothesis is".format(i+1),h)
```

Teacher's Signature

Dataset

Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Output

[['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'], ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'], ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'], ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']]

Steps of candidate Algorithm :

['Sunny', 'Warm', '?', 'Strong', '?', '?']

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Date

Expt. No. 02 Page No. 06

LAB PROGRAM-2

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

```
Code: import csv
with open("e.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)
print(data)
s = data[1][:-1]
g = [['?' for i in range(len(s))] for j in range(len(s))]
for i in data:
    if i[-1] == "Yes":
        for j in range(len(s)):
            if i[j] != s[j]:
                s[j] = '?'
                g[j][j] = '?'
    elif i[-1] == "No":
        for j in range(len(s)):
            if i[j] != s[j]:
                g[j][j] = s[j]
            else:
                g[j][j] = "?"
print("\n Steps of Candidate Elimination Algorithm", data.index(i)+1)
print(s)
print(g)
```

Teacher's Signature

Final specific hypothesis:

['sunny', 'warm', '?', 'strong', '?', '?']

Final general hypothesis:

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

Date

Expt. No. 02

Page No. 09

```
gh=[]
for i in g:
    for j in i:
        if j=='?':
            gh.append(i)
            break
print("\n Final specific hypothesis:\n",s)
print("\n Final general hypothesis:\n",gh)
```

Teacher's Signature

Dataset

Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Date

Expt. No. 03

Page No. 10

LAB PROGRAM-3

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

```
import pandas as pd
from pandas import DataFrame
df_tennis = pd.read_csv('Tennis.csv')
attribute_names = list(df_tennis.columns)
attribute_names.remove('PlayTennis')
print(attribute_names)
def entropy_of_list(list):
    from collections import Counter
    cnt = Counter(x for x in list)
    num_instances = len(list)*1.
    probs = [x/num_instances for x in cnt.values()]
    return entropy(probs)
def entropy(probs):
    import math
    return sum([-prob*math.log(prob,2) for prob in probs])
total_entropy = entropy_of_list(df_tennis['PlayTennis'])
def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
    df_split = df.groupby(split_attribute_name)
    nobs = len(df.index)*1.
    df_agg_ent = df_split.agg({target_attribute_name:[entropy_of_list, lambda x:len(x)/nobs]})
    df_agg_ent.columns=['Entropy', 'propObservations']
    new_entropy = sum(df_agg_ent['Entropy'] * df_agg_ent['propObservations'])
```

Teacher's Signature

Output

['Outlook', 'Temperature', 'Humidity', 'Wind']

Outlook IG : 0.246749819774431

Temperature IG: 0.029222565658954647

Humidity IG: 0.15183550136234136

Wind IG: 0.048127030480926927

Temperature IG: 0.5709505944546686

Humidity IG : 0.5709505944546686

Wind IG: 0.09749505944546686

Temperature IG: 0.5709505944546686

Humidity IG: 0.5709505944546686

Wind IG: 0.09749505944546686

The Resultant Decision Tree is:

{'Outlook': {'Overcast': 'Yes',
'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}

Date

Expt. No. 03

Page No. 12

```
    old_entropy = entropy_of_list(df[target_attribute_name])
    print(split_attribute_name, 'IG :', old_entropy-new_entropy)
    return old_entropy-new_entropy
def id3(df, target_attribute_name, attribute_names, default_class=None):
    from collections import Counter
    count = Counter(x for x in df[target_attribute_name])
    if len(count)==1:
        return next(iter(count))
    elif df.empty or (not attribute_names):
        return default_class
    else:
        default_class = max(count.keys())
        gain = [information_gain(df, attr, target_attribute_name) for attr in attribute_names
                ]
        print()
        index_of_max = gain.index(max(gain))
        best_attr = attribute_names[index_of_max]
        tree = {best_attr:{}}
        remaining_attribute_names = [i for i in attribute_names if i != best_attr]
        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset, target_attribute_name, remaining_attribute_names,
                          default_class)
            tree[best_attr][attr_val] = subtree
        return tree
from pprint import pprint
tree = id3(df_tennis, 'PlayTennis', attribute_names)
print("\n\nThe Resultant Decision Tree is:\n")
pprint(tree)
```

Teacher's Signature

Output

>epoch=0, lrate=0.500, error=6.350
>epoch=1, lrate=0.500, error=5.531
>epoch=2, lrate=0.500, error=5.221
>epoch=3, lrate=0.500, error=4.951
>epoch=4, lrate=0.500, error=4.519
>epoch=5, lrate=0.500, error=4.173
>epoch=6, lrate=0.500, error=3.835
>epoch=7, lrate=0.500, error=3.506
>epoch=8, lrate=0.500, error=3.192
>epoch=9, lrate=0.500, error=2.898
>epoch=10, lrate=0.500, error=2.636
>epoch=11, lrate=0.500, error=2.377
>epoch=12, lrate=0.500, error=2.153
>epoch=13, lrate=0.500, error=1.953
>epoch=14, lrate=0.500, error=1.774
>epoch=15, lrate=0.500, error=1.614
>epoch=16, lrate=0.500, error=1.472

Expt. No. 04 Date

Page No. 14

LAB PROGRAM-4

4) Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate datasets

Soln

```
from math import exp
from random import seed
from random import random
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights':[random() for i in range(n_inputs+1)]} for i in
                    range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{'weights':[random() for i in range(n_hidden+1)]} for i in
                    range(n_outputs)]
    network.append(output_layer)
    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 = []
```

Teacher's Signature

```
>epoch=17, lrate=0.500, error= 1.346
>epoch=18, lrate=0.500, error= 1.233
>epoch=19, lrate= 0.500, error= 1.132

[{'weights': [-1.4688375695432327, 1.850887325439514, 1.0858178629550297],
'output': 0.029980305604426185, 'delta': -0.0059546604162323605}, {'weights':
[0.37711098142462187, -0.0625909894552989, 0.2765723702642116], 'output':
0.9456229000211923, 'delta': 0.0026279652950863837}]

[{'weights': [2.515394649397849, -0.3391927502445985, -0.9671565426390275], 'output':
0.23648794202357787, 'delta': -0.04270059278364587}, {'weights': [-2.5584149848,
1.0036422106209202, 0.42383086467582715], 'output': 0.7790535802438647, 'delta':
0.03803132596437354}]
```



```
        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_propagate_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:
```

Date

Expt. No. 04

Page No. 19

```
            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']
def train_network(network, train, l_rate, n_epoch, n_outputs):
    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_propagate_error(network, expected)
            update_weights(network, row, l_rate)
        print('>epoch=%d, lrate=%.3f, error=%.3f' %(epoch, l_rate, sum_error))
seed(1)
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]]
n_inputs = len(dataset[0])-1
n_outputs = len(set([row[-1] for row in dataset]))
```

Teacher's Signature

Date

Expt. No. 04

Page No. 30

```
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
    print(layer)
```

Teacher's Signature

Dataset

6	148	72	25	0	28.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1

Output

Pima Indian Diabetes Dataset loaded...

Total instances available : 768

Total attributes present : 8

First five instances of dataset:

Date

Expt. No. 05

Page No. 28

LAB PROGRAM-5

5) 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.

Soln

```
import csv, random, math
import statistics as st
def loadCsv(filename):
    lines = csv.reader(open(filename,"r"))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset
def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio);
    trainSet = list(dataset)
    testSet = []
    while len(testSet) < trainSize:
        index = random.randrange(len(trainSet));
        testSet.append(trainSet.pop(index))
    return [trainSet, testSet]
def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        x = dataset[i]
        if(x[-1] not in separated):
            separated[x[-1]] = []
```

Teacher's Signature

1: [4.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]

2: [1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0, 0.0]

3: [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0, 1.0]

4: [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0, 0.0]

5: [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0, 1.0]

Dataset is split into training and testing set.

Training examples = 615

Testing examples = 153

Accuracy of the Naive Bayesian classifier is : 0.6535947712418301

Date

Expt. No. 05

Page No. 24

```
            separated[x[-1]].append(x)
        return separated
    def compute_mean_std(dataset):
        mean_std = [(st.mean(attribute), st.stdev(attribute)) for attribute in zip(*dataset)]
        del mean_std[-1]
        return mean_std
    def summarizeByClass(dataset):
        separated = separateByClass(dataset);
        summary = {}
        for classValue, instances in separated.items():
            summary[classValue] = compute_mean_std(instances)
        return summary
    def calculateProbability(x, mean, stdev):
        exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
        return (1/(math.sqrt(2*math.pi)*stdev))*exponent
    def calculateClassProbabilities(summaries, testVector):
        p = {}
        for classValue, classSummaries in summaries.items():
            p[classValue] = 1
            for i in range((len(classSummaries))):
                mean, stdev = classSummaries[i]
                x = testVector[i]
                p[classValue] *= calculateProbability(x, mean, stdev);
        return p
    def predict(summaries, testVector):
        all_p = calculateClassProbabilities(summaries, testVector)
        bestLabel, bestProb = None, -1
        for lbl, p in all_p.items():
            if bestLabel is None or p > bestProb:
```

Teacher's Signature

Date

Expt. No. 05

Page No. 06

```
                bestProb=p
                bestLabel=lbl
        return bestLabel
def perform_classification(summaries,testSet):
    predictions=[]
    for i in range(len(testSet)):
        result=predict(summaries,testSet[i])
        predictions.append(result)
    return predictions
def getAccuracy(testSet,predictions):
    correct=0
    for i in range(len(testSet)):
        if testSet[i][-1]==predictions[i]:
            correct+=1
    return (correct/float(len(testSet)))*100.0
dataset=load(csv:dataset.csv)
print('Para Indian Diabetes dataset loaded...')
print('Total instances available:',len(dataset))
print('Total attributes present:',len(dataset[0])-1)
print("First five instances of dataset:")
for i in range(5):
    print(i+1,':',dataset[i])
splitRatio=0.6
trainingset, testset=splitDataset(dataset, splitRatio)
print('\n dataset is split into training and testing set.')
print('Training examples={0}\n Testing examples={1}'.format(len(trainingset),len(testset)))
summaries=summarizeByClass(trainingset);
predictions=perform_classification(summaries,testset)
accuracy=getAccuracy(testset,predictions)
print('\n Accuracy of the Naive Bayesian Classifier is:',accuracy)
```

Teacher's Signature _____

Dataset

I love this sandwich, pos

This is an amazing place, pos

I feel very good about these beers, pos

This is my best work, pos

What an awesome view, pos

I do not like this restaurant, neg

I am tired of this stuff, neg

I can't deal with this, neg

He is my sworn enemy, neg

My boss is horrible, neg

This is an awesome place, pos

I do not like the taste of this juice, neg

I love to dance, pos

Date

Expt. No. 06 Page No. 29

LAB PROGRAM - 6

a) Assuming a set of documents that need to be classified, use the naive Bayesian classifier model to perform this task. Built in Java classes/API can be used to write the program. Calculate the accuracy, precision and recall for your dataset.

Soln

```
import pandas as pd
msg = pd.read_csv('document.csv', names=['message', 'label'])
print('Total instances in the dataset:', msg.shape[0])
msg['labelnum'] = msg.label.map({'pos':1, 'neg':0})
X = msg.message
y = msg.labelnum
print('\n The message and its label of first 5 instances are listed below:')
X5, y5 = X[0:5], msg.label[0:5]
for x, y in zip(X5, y5):
    print(x, ',', y)
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)
print('\n Dataset is split into Training and Testing samples')
print('Total training instances:', Xtrain.shape[0])
print('Total testing instances:', Xtest.shape[0])
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
Xtrain_dtm = count_vect.fit_transform(Xtrain)
Xtest_dtm = count_vect.transform(Xtest)
print('\n Total features extracted using CountVectorizer:', Xtrain_dtm.shape[1])
print('\n Features for first 5 training instances are listed below')
df = pd.DataFrame(Xtrain_dtm.toarray(), columns=count_vect.get_feature_names())
print(df[0:5])
```

Teacher's Signature

I am sick and tired of this place, neg

what a great holiday, pos

That is a bad locality to stay, neg

We will have good fun tomorrow, pos

I went to my enemy's house today, neg

Total instances in the dataset: 18

The message and its label of first 5 instances are listed below

I love this sandwich, pos

This is an amazing place, pos

I feel very good about these beers, pos

This is my best work, pos

What an awesome view, pos

Dataset is split into Training and Testing samples

Date

Expt. No. 06

Page No. 30

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm, ytrain)
predicted = clf.predict(xtest_dtm)
print('\n Classification results of testing samples are given below')
for doc,p in zip(xtest,predicted):
    pred='pos' if p==1 else 'neg'
    print("%s->%s" %(doc,pred))
from sklearn import metrics
print('\n accuracy metrics')
print('Accuracy of the classifier is', metrics.accuracy_score(ytest,predicted))
print('Recall:', metrics.recall_score(ytest,predicted), '\nPrecision:', metrics.precision_score
                                                                    (ytest,predicted))
print('\nconfusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
```

Teacher's Signature

Total training instances : 13

Total testing instances : 5

Total features extracted using CountVectorizer : 45

Features for first 5 training instances are listed below

	about	an	awesome	bad	beers	best	boss	can	deal	...	today
0	0	0	0	0	0	0	0	0	0	...	0
1	1	0	0	0	1	0	0	0	0	...	0
2	0	0	0	0	0	0	0	1	0	...	0
3	0	0	0	0	0	0	1	0	0	...	0
4	0	1	1	0	0	0	0	0	0	...	0

	tomorrow	very	view	we	went	what	will	with	work
0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0

[5 rows x 45 columns]

Classification results of testing samples are given below.

I am tired of this stuff → neg

I am sick and tired of this place → neg

Date

Expt. No. 06

Page No. 32

Teacher's Signature

That a great holiday → pos

He is my sworn enemy → neg

This is an amazing place → pos

Aaurary metrics

Accuracy of the classifier is 1.0

Recall : 1.0

Precision : 1.0

Confusion matrix

[[3 0]
[0 2]]

Expt. No. 06

Date

Page No. 34

Teacher's Signature

Output

Iris Data set loaded

Dataset is split into training and testing...

Size of training data and its label (135,4) (135,)

Size of testing data and its label (15,4) (15,)

label 0 - setosa

Label 1 - versicolor

Label 2 - virginica

Results of classification using k-nn with k=1

Sample : [4.9 3.1 1.5 0.1] Actual label : 0 Predicted label : 0

Sample: [5.4 3.9 1.7 0.4] Actual label : 0 Predicted label : 0

Sample: [6.7 3. 5. 1.7] Actual label: 1 Predicted label: 1

Sample: [4.8 3. 1.4 0.3] Actual label : 0 Predicted label : 0

Sample: [5.9 3.2 4.8 1.8] Actual label: 1 Predicted label: 2

Sample : [4.8 3. 1.4 0.1] Actual label : 0 Predicted label : 0

Sample: [6. 2.7 5.1 1.6] Actual label : 1 Predicted label : 2

Sample: [4.6 3.4 1.4 0.3] Actual label : 0 Predicted label : 0

Sample: [6. 2.2 5. 1.5] Actual label: 2 Predicted label: 1

Sample [5.5 2.3 4. 1.3] Actual label: 1 Predicted label: 1

Date

Expt. No. 09 Page No. 36

LAB PROGRAM-9

a) Write a program to implement k-Nearest Neighbours algorithm to classify the Iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Soln

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
iris = datasets.load_iris()
print("Iris Data set loaded")
x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.1)
print("Dataset is split into training and testing...")
print("Size of training data and its label", x_train.shape, y_train.shape)
print("Size of testing data and its label", x_test.shape, y_test.shape)
for i in range(len(iris.target_names)):
    print("label", i, "-", str(iris.target_names[i]))
classifier = kNeighborsClassifier(n_neighbors=1)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print("Results of classification using k-nn with k=1")
for x in range(0,len(x_test)):
    print("Sample:", str(x_test[x]), "Actual label:", str(y_test[x]), "Predicted label:",
    str(y_pred[x]))
print("Classification Accuracy:", classifier.score(x_test,y_test));
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
print('Confusion matrix')
print(confusion_matrix(y_test,y_pred))
```

Teacher's Signature

Sample : [7. 3.2 4.7 1.4] Actual label: 1 Predicted label: 1

Sample : [6.3 2.7 4.9 1.8] Actual label: 2 Predicted label: 2

Sample : [6.4 2.8 5.6 2.2] Actual label: 2 Predicted label: 2

Sample : [6. 3.4 4.5 1.6] Actual label: 1 Predicted label: 1

Sample : [6.4 3.2 4.5 1.5] Actual label: 1 Predicted label: 1

Classification Accuracy : 0.8

Confusion matrix

[[5 0 0]
 [0 5 2]
 [0 1 2]]

Accuracy Metrics

	precision	recall	f1-score	support
0	1.00	1.00	1.00	5
1	0.83	0.71	0.77	7
2	0.50	0.67	0.57	3
avg / total	0.82	0.80	0.81	15

correct prediction 0.8
wrong prediction 0.19999999999999996

Date

Expt. No. 09 Page No. 39

print('Accuracy Metrics')
print(classification_report(y_test, y_pred))
print("correct prediction", accuracy_score(y_test, y_pred))
print("Wrong prediction", 1-accuracy_score(y_test, y_pred))

Teacher's Signature