MACHINE LEARNING LABORATORY MANUAL

15CSL76

Department of Information Science & Engineering
Sahyadri College of Engineering & Management
Mangaluru

Authors : Praahas Amin & Madhura N. Hegde
Department of Information Science & Engineering
Sahyadri College of Engineering & Management
Mangaluru

PREFACE

This Machine Learning Laboratory Manual is designed for Information Science & Engineering students, Sahyadri College of Engineering and Management, Mangaluru.

Machine Learning Algorithms have gained wide application in many domains. Therefore, this laboratory manual is prepared with the intention of providing a learning opportunity for the students to understand the basics of programming Machine Learning Algorithms in Python

The lab experiments enhance the learning curve for the students and give them hands-on experience on the different concepts that were discussed in the theory classes.

The manual contains the prescribed experiments for easy & quick understanding of the students. The authors have gathered materials from Books, Journals and Web resources.

We hope that this practical manual will be helpful for the students of Information Science & Engineering for understanding the subject from the point of view of applied aspects.

There is always scope for improvement in the manual. We would appreciate to receive valuable suggestions from readers and users for future use.

This Laboratory manual is based on the syllabus provided by VTU under the subject code 15CSL76.

ACKNOWLEDGMENTS

We are highly indebted to **Dr. Shamanth Rai.** Professor & H.O.D. Department of Information Science & Engineering for providing us the opportunity to prepare this manual as well as for his guidance and constant supervision.

We would like to express our special gratitude and thanks to **Dr. Srinivas Rao Kunte**, Principal Sahyadri College of Engineering & Management, Mangaluru.

We are also thankful to **Dr. D. L. Prabhakara**, Director Sahyadri Group of Institutions, and **Dr. Manjunath Bhandary**, Chairman, Sahyadri Group of Institutions.

We are also thankful to **Mr. Praveen Damle** for proofreading and constant support in the Laboratories.

We are grateful to **Mr. Gautham G. Rao (4SF15IS028)** for all the support provided in development of the code.

We are thankful to the students and colleagues for their support and encouragement.

Praahas Amin
Madhura N. Hegde

S.No.	Experiments	Page
1	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.	4
2	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.	6
3	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.	8
4	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.	11
5	Write a program to implement the Naïve Bayes classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.	13
6	Assuming a set of documents that need to be classified, use the naïve 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 data set.	16
7	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. You can use Java/Python ML library classes/API.	19
8	Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.	23
9	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.	26
10	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	28

All laboratory experiments are to be included for practical examination.

- Students are allowed to pick one experiment from the lot.
- Strictly follow the instructions as printed on the cover page of answer script
- Marks distribution: Procedure + Conduction + Viva:**20 + 50 +10 (80)**

Change of experiment is allowed only once and marks allotted to the procedure part to be made 0.

Subject Code: 15CSL76 | IA Marks: 20 | Exam Marks: 80 | Exam Hours: 03 | Credits: 02

Experiment 1: Find-S Algorithm

Aim: 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.

```
import pandas as pd
from pandas import DataFrame
data = DataFrame.from_csv('EnjoySport.csv')
columnLength= data.shape[1]
print (data.values)
h = ['0']*(columnLength-1)
hp=[]
hn=[]
for trainingExample in data.values:
    if trainingExample[-1]!='no':
        hp.append(list(trainingExample))
    else:
        hn.append(list(trainingExample))
for i in range (len(hp)):
    for j in range(columnLength-1):
        if (h[j]=='0'):
            h[j]=hp[i][j]
        if (h[j]!=hp[i][j]):
            h[j]='?'
        else:
            h[j]=hp[i][j]
print('\nThe positive Hypotheses are:',hp)
print('\nThe negative Hypotheses are:',hn)
print('\nThe Maximally Specific Hypothesis h is:',h)
```

Output:

```
The positive Hypotheses are
[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'],
 ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'],
 ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]

The negative Hypotheses are
[['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']]

The Maximally Specific Hypothesis h is
['sunny', 'warm', '?', 'strong', '?', '?']
```

Dataset:

Sl.No	sky	airTemp	humidity	wind	water	forecast	enjoySport
0	sunny	warm	normal	strong	warm	same	yes
1	sunny	warm	high	strong	warm	same	yes
2	rainy	cold	high	strong	warm	change	no
3	sunny	warm	high	strong	cool	change	yes

Source: https://github.com/praahas/machine-learning-vtu

Experiment 2: Candidate-Elimination Algorithm

Aim: 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.

```
from pandas import DataFrame
data=DataFrame.from_csv('EnjoySport.csv')
concepts=data.values[:,:-1]
target=data.values[:,-1]

def learn(concepts, target):
    specific_h = concepts[0].copy()
    general_h = [['?' for i in range(len(specific_h))] for i in range(len(specific_h))]
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            #print(target[i])
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
    indices = [i for i,val in enumerate(general_h) if val==['?' for i in range(len(specific_h))]]
    for i in indices:
        general_h.remove(['?' for i in range(len(specific_h))])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)
print("Final S:", s_final)
print("Final G:", g_final)
```

Output:

```
Final S: ['sunny' 'warm' '?' 'strong' '?' '?']
Final G: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Dataset:

```
          sky airTemp humidity    wind water forecast enjoySport
Sl.No
0       sunny    warm   normal  strong  warm     same        yes
1       sunny    warm     high  strong  warm     same        yes
2       rainy    cold     high  strong  warm   change         no
3       sunny    warm     high  strong  cool   change        yes
```

Source: https://github.com/ggrao1/Candidate-Elimination

Experiment 3: Decision Tree based ID3 Algorithm

Aim: 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.

```
def infoGain(P, N):
    import math
    return -P / (P + N) * math.log2(P / ( P + N)) - N / (P + N) * math.log2(N / (P + N))

def insertNode(tree, addTo, Node):
    for k, v in tree.items():
        if isinstance(v, dict):
            tree[k] = insertNode(v, addTo, Node)
    if addTo in tree:
        if isinstance(tree[addTo], dict):
            tree[addTo][Node] = 'None'
        else:
            tree[addTo] = {Node:'None'}
    return tree

def insertConcept(tree, addTo, Node):
    for k, v in tree.items():
        if isinstance(v, dict):
            tree[k] = insertConcept(v, addTo, Node)
    if addTo in tree:
        tree[addTo] = Node
    return tree

def getNextNode(data, AttributeList, concept, conceptVals, tree, addTo):
    Total = data.shape[0]
    if Total == 0:
        return tree

    countC = {}
    for cVal in conceptVals:
        dataCC = data[data[concept] = = cVal]
        countC[cVal] = dataCC.shape[0]

    if countC[conceptVals[0]] = = 0:
        tree = insertConcept(tree, addTo, conceptVals[1])
        return tree

    if countC[conceptVals[1]] = = 0:
        tree = insertConcept(tree, addTo, conceptVals[0])
        return tree

    ClassEntropy = infoGain(countC[conceptVals[1]],countC[conceptVals[0]])
    Attr = {}
    for a in AttributeList:
        Attr[a] = list(set(data[a]))
    AttrCount = {}
    EntropyAttr = {}
```

```
    for att in Attr:
        for vals in Attr [att]:
            for c in conceptVals:
                iData = data[data[att] = = vals]
                dataAtt = iData[iData[concept] = = c]
                AttrCount[c] = dataAtt.shape[0]
            TotalInfo = AttrCount[conceptVals[1]] + AttrCount[conceptVals[0]]
            if AttrCount[conceptVals[1]] = = 0 or AttrCount[conceptVals[0]] = = 0:
                InfoGain=0
            else:
                InfoGain = infoGain(AttrCount[conceptVals[1]], AttrCount[conceptVals[0]])

            if att not in EntropyAttr:
                EntropyAttr[att] = ( TotalInfo / Total ) * InfoGain
            else:
                EntropyAttr[att] = EntropyAttr[att] + ( TotalInfo / Total ) * InfoGain

    Gain = {}
    for g in EntropyAttr:
        Gain[g] = ClassEntropy - EntropyAttr[g]

    Node = max(Gain, key = Gain.get)
    tree = insertNode(tree, addTo, Node)
    for nD in Attr[Node]:
        tree = insertNode(tree, Node, nD)
        newData = data[data[Node] = = nD].drop(Node, axis = 1)
        AttributeList=list(newData)[:-1]   #New Attribute List
        tree = getNextNode(newData, AttributeList, concept, conceptVals, tree, nD)
    return tree

def main():
    from pandas import DataFrame
    data = DataFrame.from_csv('PlayTennis.csv')
    print(data)
    AttributeList = list(data)[:-1]
    concept = str(list(data)[-1])
    conceptVals = list(set(data[concept]))
    tree = getNextNode(data, AttributeList, concept, conceptVals, {'root':'None'}, 'root')
    print(tree)

main()
```

Output:

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

Dataset:

slno	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

Source: https://github.com/ggrao1/decision-tree

Experiment 4: Artificial Neural Network using Backpropagation Algorithm

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

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0)
y = y/100

def sigmoid (x):
    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):
    return x * (1 - x)

epoch=7000
learning_rate=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wo=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bo=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
    net_h=np.dot(X,wh) + bh
    sigma_h= sigmoid(net_h)
    net_o= np.dot(sigma_h,wo)+ bo
    output = sigmoid(net_o)
    deltaK = (y-output)* derivatives_sigmoid(output)
    deltaH =  deltaK.dot(wo.T) * derivatives_sigmoid(sigma_h)
    wo = wo + sigma_h.T.dot(deltaK) *learning_rate
    wh = wh + X.T.dot(deltaH) *learning_rate

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89454606]
 [0.88145161]
 [0.89393293]]
```

Source: https://github.com/praahas/machine-learning-vtu

Experiment 5: Naïve Bayes Classifier

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

```
def probAttr(data,attr,val):
    Total=data.shape[0]
    cnt = len(data[data[attr] == val])
    return cnt,cnt/Total

def train(data,Attr,conceptVals,concept):
    conceptProbs = {}
    countConcept={}
    for cVal in conceptVals:
        countConcept[cVal],conceptProbs[cVal] = probAttr(data,concept,cVal)
    AttrConcept = {}
    probability_list = {}
    for att in Attr: #Create a tree for attribute
        AttrConcept[att] = {}
        probability_list[att] = {}
        for val in Attr[att]:
            AttrConcept[att][val] = {}
            a,probability_list[att][val] = probAttr(data,att,val)
            for cVal in conceptVals:
                dataTemp = data[data[att]==val]
                AttrConcept[att][val][cVal] = len(dataTemp[dataTemp[concept] == cVal])/countConcept[cVal]

    print("P(A) : ",conceptProbs,"\n")
    print("P(X/A) : ",AttrConcept,"\n")
    print("P(X) : ",probability_list,"\n")
    return conceptProbs,AttrConcept,probability_list

def test(examples,Attr,concept_list,conceptProbs,AttrConcept,probability_list):
    misclassification_count=0
    Total = len(examples)
    for ex in examples:
        px={}
        for a in Attr:
            for x in ex:
                for c in concept_list:
                    if x in AttrConcept[a]:
                        if c not in px:
                            px[c] = conceptProbs[c]*AttrConcept[a][x][c]/probability_list[a][x]
                        else:
                            px[c] = px[c]*AttrConcept[a][x][c]/probability_list[a][x]
        print(px)
        classification = max(px,key=px.get)
        print("Classification :",classification,"Expected :",ex[-1])
        if(classification!=ex[-1]):
            misclassification_count+=1
    misclassification_rate=misclassification_count*100/Total
    accuracy=100-misclassification_rate
```

```
    print("Misclassification Count={}".format(misclassification_count))
    print("Misclassification Rate={}%".format(misclassification_rate))
    print("Accuracy={}%".format(accuracy))

def main():
    import pandas as pd
    from pandas import DataFrame
    data = DataFrame.from_csv('PlayTennis_train1.csv')
    concept=str(list(data)[-1])
    concept_list = set(data[concept])
    Attr={}
    for a in list(data)[:-1]:
        Attr[a] = set(data[a])
    conceptProbs,AttrConcept,probability_list = train(data,Attr,concept_list,concept)

    examples = DataFrame.from_csv(PlayTennis_test1.csv')
    test(examples.values,Attr,concept_list,conceptProbs,AttrConcept,probability_list)

main()
```

Output:

```
P(A) :  {'No': 0.35714285714285715, 'Yes': 0.6428571428571429}

P(X/A) :  {'Outlook': {'Overcast': {'No': 0.0, 'Yes': 0.4444444444444444}, 'Sunny': {'No': 0.6, 'Yes':
0.2222222222222222}, 'Rain': {'No': 0.4, 'Yes': 0.3333333333333333}}, 'Temperature': {'Hot': {'No': 0.
4, 'Yes': 0.2222222222222222}, 'Mild': {'No': 0.4, 'Yes': 0.4444444444444444}, 'Cool': {'No': 0.2, 'Ye
s': 0.3333333333333333}}, 'Humidity': {'High': {'No': 0.8, 'Yes': 0.3333333333333333}, 'Normal': {'No':
0.2, 'Yes': 0.6666666666666666}}, 'Wind': {'Strong': {'No': 0.6, 'Yes': 0.3333333333333333}, 'Weak':
{'No': 0.4, 'Yes': 0.6666666666666666}}}

P(X) :  {'Outlook': {'Overcast': 0.2857142857142857, 'Sunny': 0.35714285714285715, 'Rain': 0.3571428571
4285715}, 'Temperature': {'Hot': 0.2857142857142857, 'Mild': 0.42857142857142855, 'Cool': 0.28571428571
42857}, 'Humidity': {'High': 0.5, 'Normal': 0.5}, 'Wind': {'Strong': 0.42857142857142855, 'Weak': 0.571
4285714285714}}

{'No': 0.9408000000000001, 'Yes': 0.24197530864197522}
Classification : No Expected : No
Misclassification Count=0
Misclassification Rate=0.0%
Accuracy=100.0%
```

Dataset:

Training Set

slno	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

Testing example

slno	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Cool	High	Strong	No

Source: https://github.com/ggrao1/NaiveBayes

Experiment 6: Naïve Bayes Classifier using API

Aim: Assuming a set of documents that need to be classified, use the naïve 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 data set.

```
import pandas as pd
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
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)
from sklearn.feature_extraction.text import CountVectorizer
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())
print(df[0:5])
from sklearn.naive_bayes import MultinomialNB
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))

from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score

print('Accuracy Metrics: \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:

```
    am  amazing  an  and  awesome  bad  best  boss  dance  do  ...   sworn  \
0    0        0   0    0        0    0     0     0      0   1  ...       0
1    0        0   1    0        1    0     0     0      0   0  ...       0
2    0        0   0    0        0    0     0     0      0   0  ...       1
3    0        0   0    0        0    0     1     0      0   0  ...       0
4    0        0   0    0        0    0     0     0      0   0  ...       0

   that  this  tired  to  today  view  went  what  work
0     0     1      0   0      0     0     0     0     0
1     0     0      0   0      0     1     0     1     0
2     0     0      0   0      0     0     0     0     0
3     0     1      0   0      0     0     0     0     1
4     0     0      0   1      1     0     1     0     0
I do not like this restaurant -> neg
What an awesome view -> pos
He is my sworn enemy -> neg
This is my best work -> pos
I went to my enemy's house today -> pos
Accuracy Metrics:
Accuracy:  0.8
Recall:  1.0
Precision:  0.666666666667
Confusion Matrix:
 [[2 1]
 [0 2]]
```

Source: https://github.com/rumaan/machine-learning-lab-vtu

Dataset:

	message	label
0	I love this sandwich	pos
1	This is an amazing place	pos
2	I feel very good about these beers	pos
3	This is my best work	pos
4	What an awesome view	pos
5	I do not like this restaurant	neg
6	I am tired of this stuff	neg
7	I can't deal with this	neg
8	He is my sworn enemy	neg
9	My boss is horrible	neg
10	This is an awesome place	pos
11	I do not like the taste of this juice	neg
12	I love to dance	pos
13	I am sick and tired of this place	neg
14	What a great holiday	pos
15	That is a bad locality to stay	neg
16	We will have good fun tomorrow	pos
17	I went to my enemy's house today	neg

Experiment 7: Bayesian Network

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. You can use Java/Python ML library classes/API.

```
from pgmpy.models import BayesianModel
cancer_model = BayesianModel([('Pollution', 'Cancer'),('Smoker', 'Cancer'),('Cancer', 'Xray'),('Cancer',
'Dyspnoea')])
cancer_model.nodes()
cancer_model.edges()
cancer_model.get_cpds()
from pgmpy.factors.discrete import TabularCPD

cpd_poll = TabularCPD(variable='Pollution', variable_card=2, values=[[0.9], [0.1]])
cpd_smoke = TabularCPD(variable='Smoker', variable_card=2, values=[[0.3], [0.7]])
cpd_cancer = TabularCPD(variable='Cancer', variable_card=2, values=[[0.03, 0.05, 0.001, 0.02],
                        [0.97, 0.95, 0.999, 0.98]],evidence=['Smoker', 'Pollution'], evidence_card=[2, 2])
cpd_xray = TabularCPD(variable='Xray', variable_card=2,  values=[[0.9, 0.2], [0.1, 0.8]],
                evidence=['Cancer'], evidence_card=[2])
cpd_dysp = TabularCPD(variable='Dyspnoea', variable_card=2, values=[[0.65, 0.3], [0.35, 0.7]],
                evidence=['Cancer'], evidence_card=[2])
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp)
cancer_model.check_model()

cancer_model.get_cpds()
print(cancer_model.get_cpds('Pollution'))
print(cancer_model.get_cpds('Smoker'))
print(cancer_model.get_cpds('Xray'))
print(cancer_model.get_cpds('Dyspnoea'))
print(cancer_model.get_cpds('Cancer'))

cancer_model.local_independencies('Xray')
cancer_model.local_independencies('Pollution')
cancer_model.local_independencies('Smoker')
cancer_model.local_independencies('Dyspnoea')
cancer_model.local_independencies('Cancer')
cancer_model.get_independencies()

from pgmpy.inference import VariableElimination
cancer_infer = VariableElimination(cancer_model)

q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1})
print(q['Cancer'])

q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1,'Pollution': 1})
print(q['Cancer'])
```

Output:

Pollution_0	0.9
Pollution_1	0.1

Smoker_0	0.3
Smoker_1	0.7

Cancer	Cancer_0	Cancer_1
Xray_0	0.9	0.2
Xray_1	0.1	0.8

Cancer	Cancer_0	Cancer_1
Dyspnoea_0	0.65	0.3
Dyspnoea_1	0.35	0.7

Smoker	Smoker_0	Smoker_0	Smoker_1	Smoker_1
Pollution	Pollution_0	Pollution_1	Pollution_0	Pollution_1
Cancer_0	0.03	0.05	0.001	0.02
Cancer_1	0.97	0.95	0.999	0.98

Inferencing:

Cancer	phi(Cancer)
Cancer_0	0.0029
Cancer_1	0.9971

Cancer	phi(Cancer)
Cancer_0	0.0200
Cancer_1	0.9800

Diagnosis of heart patients using standard Heart Disease Data Set:

```
import numpy as np
from urllib.request import urlopen
import urllib
import matplotlib.pyplot as plt # Visuals
import seaborn as sns
import sklearn as skl
import pandas as pd

Cleveland_data_URL = 'http://archive.ics.uci.edu/ml/machine-learning-databases/heart-
disease/processed.hungarian.data'
np.set_printoptions(threshold=np.nan) #see a whole array when we output it

names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal',
'heartdisease']
heartDisease = pd.read_csv(urlopen(Cleveland_data_URL), names = names) #gets Cleveland data

del heartDisease['ca']
del heartDisease['slope']
del heartDisease['thal']
del heartDisease['oldpeak']

heartDisease = heartDisease.replace('?', np.nan)

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

model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),
                      ('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),
                     ('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])

# Learing CPDs using Maximum Likelihood Estimators
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

print(model.get_cpds('age'))
print(model.get_cpds('chol'))
print(model.get_cpds('sex'))
model.get_independencies()

from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q['heartdisease'])

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'chol': 100})
print(q['heartdisease'])
```

Output:

Diagnosis:

heartdisease	phi(heartdisease)
heartdisease_0	0.6333
heartdisease_1	0.3667

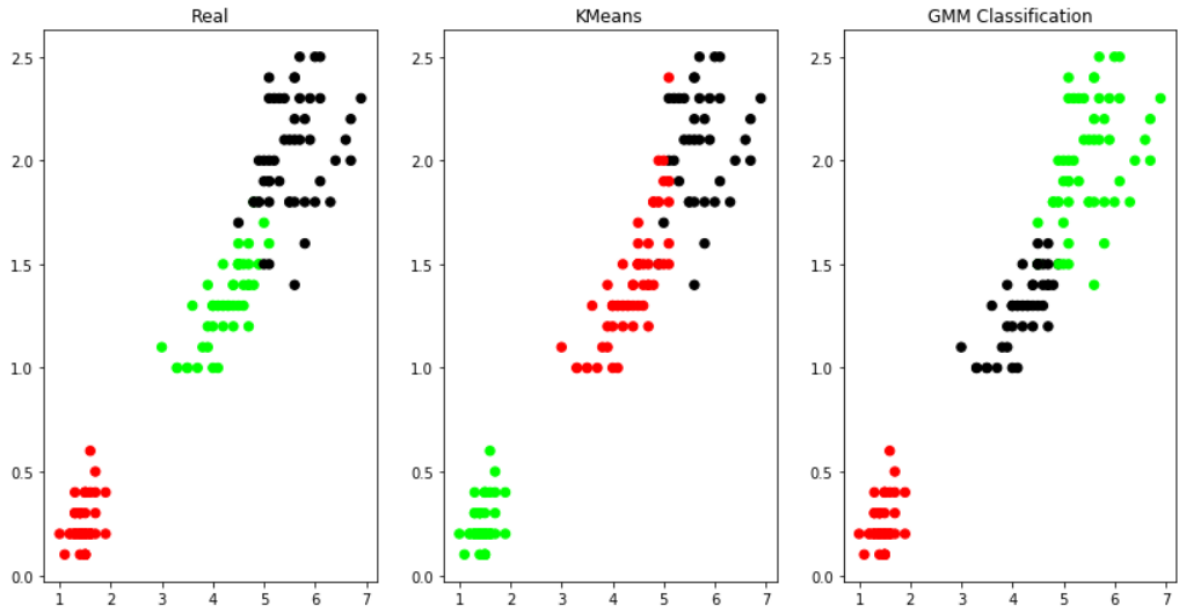
heartdisease	phi(heartdisease)
heartdisease_0	1.0000
heartdisease_1	0.0000

Experiment 8: Clustering using EM Algorithm & k-Means Algorithm

Aim: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset=load_iris()
X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])

#REAL PLOT
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')

#KMeans -PLOT
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')

#GMM PLOT
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

Output



Dataset:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2
11	4.8	3.4	1.6	0.2
12	4.8	3.0	1.4	0.1
13	4.3	3.0	1.1	0.1
14	5.8	4.0	1.2	0.2
15	5.7	4.4	1.5	0.4
16	5.4	3.9	1.3	0.4
17	5.1	3.5	1.4	0.3
18	5.7	3.8	1.7	0.3
19	5.1	3.8	1.5	0.3
20	5.4	3.4	1.7	0.2
21	5.1	3.7	1.5	0.4
22	4.6	3.6	1.0	0.2
23	5.1	3.3	1.7	0.5
24	4.8	3.4	1.9	0.2
25	5.0	3.0	1.6	0.2
26	5.0	3.4	1.6	0.4
27	5.2	3.5	1.5	0.2
28	5.2	3.4	1.4	0.2
29	4.7	3.2	1.6	0.2
..	...	...	...	...
136	6.3	3.4	5.6	2.4
137	6.4	3.1	5.5	1.8
138	6.0	3.0	4.8	1.8
139	6.9	3.1	5.4	2.1
140	6.7	3.1	5.6	2.4
141	6.9	3.1	5.1	2.3
142	5.8	2.7	5.1	1.9
143	6.8	3.2	5.9	2.3
144	6.7	3.3	5.7	2.5
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

Experiment 9: k-Nearest Neighbour Algorithm

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

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
dataset=load_iris()
X_train,X_test,y_train,y_test=train_test_split(dataset["data"],dataset["target"],random_state=0)
clf=KNeighborsClassifier(n_neighbors=1)
clf.fit(X_train,y_train)
for i in range(len(X_test)):
   x=X_test[i]
   x_new=np.array([x])
   prediction=clf.predict(x_new)

print("TARGET=",y_test[i],dataset["target_names"][y_test[i]],"PREDICTED=",prediction,dataset["target_
names"][prediction])
print(clf.score(X_test,y_test))
```

Output:

```
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
```

```
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 2 virginica PREDICTED= [2] ['virginica']
TARGET= 1 versicolor PREDICTED= [1] ['versicolor']
TARGET= 0 setosa PREDICTED= [0] ['setosa']
TARGET= 1 versicolor PREDICTED= [2] ['virginica']
```

0.973684210526

Dataset:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2
11	4.8	3.4	1.6	0.2
12	4.8	3.0	1.4	0.1
13	4.3	3.0	1.1	0.1
14	5.8	4.0	1.2	0.2
15	5.7	4.4	1.5	0.4
16	5.4	3.9	1.3	0.4
17	5.1	3.5	1.4	0.3
18	5.7	3.8	1.7	0.3
19	5.1	3.8	1.5	0.3
20	5.4	3.4	1.7	0.2
21	5.1	3.7	1.5	0.4
22	4.6	3.6	1.0	0.2
23	5.1	3.3	1.7	0.5
24	4.8	3.4	1.9	0.2
25	5.0	3.0	1.6	0.2
26	5.0	3.4	1.6	0.4
27	5.2	3.5	1.5	0.2
28	5.2	3.4	1.4	0.2
29	4.7	3.2	1.6	0.2
..	...	...	...	...
136	6.3	3.4	5.6	2.4
137	6.4	3.1	5.5	1.8
138	6.0	3.0	4.8	1.8
139	6.9	3.1	5.4	2.1
140	6.7	3.1	5.6	2.4
141	6.9	3.1	5.1	2.3
142	5.8	2.7	5.1	1.9
143	6.8	3.2	5.9	2.3
144	6.7	3.3	5.7	2.5
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

Source: https://github.com/praahas/machine-learning-vtu

Experiment 10: Locally Weighted Regression Algorithm

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

```
from math import ceil
import numpy as np
from scipy import linalg

def lowess(x, y, f, iterations):
    n = len(x)
    r = int(ceil(f * n))
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
    w = (1 - w ** 3) ** 3
    yest = np.zeros(n)
    delta = np.ones(n)
    for iteration in range(iterations):
        for i in range(n):
            weights = delta * w[:, i]
            b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
            A = np.array([[np.sum(weights), np.sum(weights * x)],[np.sum(weights * x), np.sum(weights * x *
x)]])
            beta = linalg.solve(A, b)
            yest[i] = beta[0] + beta[1] * x[i]

        residuals = y - yest
        s = np.median(np.abs(residuals))
        delta = np.clip(residuals / (6.0 * s), -1, 1)
        delta = (1 - delta ** 2) ** 2

    return yest

def main():
    import math
    n = 100
    x = np.linspace(0, 2 * math.pi, n)
    y = np.sin(x) + 0.3 * np.random.randn(n)
    f =0.25
    iterations=3
    yest = lowess(x, y, f, iterations)

    import matplotlib.pyplot as plt
    plt.plot(x,y,"r.")
    plt.plot(x,yest,"b-")

main()
```

Output: