Machine Learning Laboratory (15CSL76)

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.

Lab1data.csv:

```
sky,air_temp,humidity,wind,water,forecast,enjoy_sport
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
```

Lab1.py:

```
import  csv
file=open('Lab1data.csv')     #Read csv data
data=list(csv.reader(file))   #Convert into list format
length=len(data[0])-1         #-1 because we don't need the target variable
h=['0']*length                #Initial hypothesis
print("Initial Hypothesis:",h)
print('Data:')
for i in data:
    print(i)
col=data.pop(0)               #Removing the column names
for i in range(len(data)):
    if data[i][length]=='yes':            #Considering only the positive examples
        for j in range (len(data[i])-1):  #Not considering the target variable
            if h[j]=='0':
                h[j]=data[i][j]           #If 0 then copy the data
            if h[j]!=data[i][j]:    #If not equal to previous hypothesis then put '?'
                h[j]='?'
print("Final Hypothesis(Most Specific):",h)     #Print hypothesis
```

Output:

```
Initial Hypothesis: ['0', '0', '0', '0', '0', '0']
Data:
['sky', 'air_temp', 'humidity', 'wind', 'water', 'forecast', 'enjoy_sport']
['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']
Final Hypothesis(Most Specific): ['sunny', 'warm', '?', 'strong', '?', '?']
```

---

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.

Lab2data.csv:

```
sky,air_temp,humidity,wind,water,forecast,enjoy_sport
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
```

Lab2.py:

```
import csv
file=open('Lab2data.csv')
data=list(csv.reader(file))[1:]
concepts=[]
target=[]
for i in data:
    concepts.append(i[:-1])
    target.append(i[-1])
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":
        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 indices:
    general_h.remove(['?','?','?','?','?','?'])
print("Final Specific:",specific_h,sep="\n")
print("Final General:",general_h,sep="\n")
```

Output:
Final Specific:
['sunny', 'warm', '?', 'strong', '?', '?']
Final General:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]

---

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.

Lab3data.csv:

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

Rainy,Cool,Normal,Strong,No
Overcast,Cool,Normal,Strong,Yes
Sunny,Mild,High,Weak,No
Sunny,Cool,Normal,Weak,Yes
Rainy,Mild,Normal,Weak,Yes
Sunny,Mild,Normal,Strong,Yes
Overcast,Mild,High,Strong,Yes
Overcast,Hot,Normal,Weak,Yes
Rainy,Mild,High,Strong,No

Lab3.py:

```
import csv
import math
def major_class(attrs,data,target):
    freq={}
    i=attrs.index(target)
    for row in data:
        freq[row[i]]=freq.get(row[i],0)+1
    return max(freq,key=freq.get)
def entropy(attrs,data,target):
    freq={}
    entropy=0
    i=len(attrs)-1
    for row in data:
        freq[row[i]]=freq.get(row[i],0)+1
    for val in freq.values():
        entropy+=(-val/len(data))*math.log(val/len(data),2)
    return entropy
def info_gain(attrs,data,attribute,target):
    freq={}
    sub_entropy=0
    i=attrs.index(attribute)
    for row in data:
        freq[row[i]]=freq.get(row[i],0)+1
    for key in freq.keys():
        prob=freq[key]/sum(freq.values())
        data_subset=[row for row in data if row[i]==key]
        sub_entropy+=prob*entropy(attrs,data_subset,target)
    data_subset=[row for row in data if row[0]!=attrs[0]]
    return (entropy(attrs,data_subset,target)-sub_entropy)
def choose_attr(data,attrs,target):
    best=attrs[0]
    max_gain=0
    for attr in attrs:
        if attr!=target:
            new_gain=info_gain(attrs,data,attr,target)
            if new_gain>max_gain:
                max_gain=new_gain
                best=attr
    return best
def get_values(data,attrs,attribute):
    i=attrs.index(attribute)
```

```
    values=[]
    values=[row[i] for row in data if row[i]!=attribute and row[i] not in values]
    return values
def get_data(data,attrs,best,val):
    i=attrs.index(best)
    new_data=[[row[j] for j in range(len(row)) if j!=i] for row in data if row[i]==val
]
    return new_data
def build_tree(data,attrs,target):
    vals=[row[attrs.index(target)] for row in data]
    default=major_class(attrs,data,target)
    if not data or (len(attrs)-1)<=0:
        return default
    elif vals.count(vals[0])==len(vals):
        return vals[0]
    else:
        best=choose_attr(data,attrs,target)
        tree={best:{}}
        for val in get_values(data,attrs,best):
            new_data=get_data(data,attrs,best,val)
            new_attrs=attrs[:]
            new_attrs.remove(best)
            subtree=build_tree(new_data,new_attrs,target)
            tree[best][val]=subtree
    return tree
def classify(attrs,inst,tree):
    attribute=next(iter(tree))
    i=attrs.index(attribute)
    if inst[i] in tree[attribute].keys():
        result=tree[attribute][inst[i]]
        if isinstance(result,dict):
            return classify(attrs,inst,result)
        else:
            return result
    else:
        return None
file=open('Lab3data.csv')
data=list(csv.reader(file))
attrs=data[0]
tree=build_tree(data[1:],attrs,attrs[-1])
print('Decistion Tree: \n',tree)
inst=input("Enter a test instance:").split(',')
print('Output Class: ',classify(attrs,inst,tree))
```

Output:
Decistion Tree:
{'Outlook': {'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}, 'Overcast': 'Yes', 'Rainy': {'Wind': {'Weak': 'Yes', 'Strong': 'No'}}}}
Enter a test instance:Sunny,Hot,High,Weak
Output Class: No

---

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

```
from random import random,seed
def initialize(ninputs,n_hidden,n_output):
    network=[]
    hidden_layer=[{'w':[random() for i in range(n_inputs+1)]} for i in range(n_hidden)
] #Initialize Weights and Biases for hidden layers
    network.append(hidden_layer)
    output_layer=[{'w':[random() for i in range(n_hidden+1)]} for i in range(n_output)
] #Initialize Weights and Biases for output layer
    network.append(output_layer)
    return network
def activate(w,i):
    activation=w[-1] #Bias
    for x in range(len(w)-1):
        activation+=w[x]*i[x] #WX
        return activation #WX+B
from math import exp
def sigmoid(a):
    return 1/(1+exp(-a))
def forward_prop(network,row):
    inputs=row
    for layer in network:
        new_inputs=[]
        for neuron in layer:
            activation=activate(neuron['w'],inputs) #Compute Activations
            neuron['output']=sigmoid(activation) #Compute Sigmoid
            new_inputs.append(neuron['output'])
        inputs=new_inputs
    return inputs
def sigmoid_derivative(output):
    return output * (1-output) #Derivative of 1/(1+e^-x)
def backprop(network,expected):
    for i in reversed(range(len(network))):
        layer=network[i]
        errors=[]
        if i!=len(network)-1: #Output Layer
            for j in range(len(layer)):
                error=0
                for neuron in network[i+1]:
                    error+=(neuron['w'][j]*neuron['delta'])
                errors.append(error)
        else:
                for j in range(len(layer)):
                    neuron=layer[j]
                    errors.append(expected[j]-neuron['output']) #Append Errors
        for j in range(len(layer)):
                neuron=layer[j]
                neuron['delta']=errors[j]*sigmoid_derivative(neuron['output']) #Comput
e Gradients
def update_weights(network,row,lrate): #Gradient Descent
    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['w'][j]+=lrate*neuron['delta']*inputs[j] #Weights
                    neuron['w'][-1]+=lrate*neuron['delta'] #Bias
def train_network(network,train,lrate,epochs,n_output):
    for epoch in range(epochs):
        sum_err=0
        for row in train:
            outputs=forward_prop(network,row)
            expected=[0 for i in range(n_output)]
            expected[row[-1]]=1
            sum_err+=sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
            backprop(network,expected)
            update_weights(network,row,lrate)
        print('epoch=%d, lrate=%.3f,error=%.3f'%(epoch,lrate,sum_err))
seed(1)
data=[[1,2,1],[2,4,1],[3,6,1],[4,8,1],[5,10,1],[1,12,0],[2,13,0],[3,16,0],[4,8,0],[5,1
0,0]]
n_inputs=len(data[0])-1
n_outputs=len(set(row[-1] for row in data))
network=initialize(n_inputs,2,n_outputs)
train_network(network,data,0.1,20,n_outputs)
for layer in network:
    print(layer)
```

Output:

```
epoch=0, lrate=0.100,error=5.356
epoch=1, lrate=0.100,error=5.287
epoch=2, lrate=0.100,error=5.233
epoch=3, lrate=0.100,error=5.193
epoch=4, lrate=0.100,error=5.163
epoch=5, lrate=0.100,error=5.140
epoch=6, lrate=0.100,error=5.124
epoch=7, lrate=0.100,error=5.112
epoch=8, lrate=0.100,error=5.103
epoch=9, lrate=0.100,error=5.096
epoch=10, lrate=0.100,error=5.091
epoch=11, lrate=0.100,error=5.088
epoch=12, lrate=0.100,error=5.086
epoch=13, lrate=0.100,error=5.084
epoch=14, lrate=0.100,error=5.082
epoch=15, lrate=0.100,error=5.081
epoch=16, lrate=0.100,error=5.081
epoch=17, lrate=0.100,error=5.080
epoch=18, lrate=0.100,error=5.080
epoch=19, lrate=0.100,error=5.080
```

[{'w': [0.13436424411240122, 0.8474337369372327, 0.763774618976614], 'output': 0.8077717153102655, 'delta': 0.011337545739369 33}, {'w': [0.2550690257394217, 0.49543508709194095, 0.4494910647887381], 'output': 0.8487507203692017, 'delta': -0.0013102220922822422}] [{'w': [0.5069777005886287, 0.6425512871970988, -0.2838839321475699], 'output': 0.5235104041880259, 'delta': 0.11885902452022241}, {'w': [-0.12290824283687743, 0.6836071005959973, 0.0354145904333385], 'output': 0.49219143815435656, 'delta': -0.12301784883596455}]

---

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.

Lab5data.csv:

```
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1 …………..
```

Lab5.py:

```
import csv,math,random,statistics
random.seed(10)
def prob(x,mean,stdev):
    e=math.exp(-(math.pow(x-mean,2))/(2*math.pow(stdev,2)))
    return (1/(math.sqrt(2*math.pi)*stdev))*e
file=open('lab5data.csv')
data=[[float(a) for a in row] for row in csv.reader(file)]
print('Size of dataset is: ',len(data))
train_indices=random.sample(range(len(data)),int(0.7*len(data)))
xtrain=[data[i] for i in train_indices]
xtest=[data[i] for i in range(len(data)) if i not in train_indices]
classes={}
for samples in xtrain:
    last=int(samples[-1])
    if last not in classes:
        classes[last]=[]
    classes[last].append(samples)
summaries={}
for classVal,traindata in classes.items():
    summary=[(statistics.mean(attr),statistics.stdev(attr)) for attr in zip(*traindata
)]
    del summary[-1]
    summaries[classVal]=summary
prediction=[]
for test in xtest:
    probs={}
    for classVal,summary in summaries.items():
        probs[classVal]=1
        for i,attr in enumerate(summary):
            probs[classVal]*=prob(test[i],attr[0],attr[1])
        bestlabel,bestprob=None,0
    for classVal,p in probs.items():
        if bestlabel is None or p>bestprob:
            bestprob=p
            bestlabel=classVal
    prediction.append(bestlabel)
```

```
correct=0
for i,key in enumerate(xtest):
    if xtest[i][-1]==prediction[i]:
        correct+=1
print("Accuracy: ",correct*100/len(xtest))
```

Output:
Size of dataset is: 768
Accuracy: 77.92207792207792

---

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.

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn import metrics
import numpy as np

categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
X_train = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, rand
om_state=42)
text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', MultinomialNB())
                     ])
text_clf.fit(X_train.data, X_train.target)
X_test = fetch_20newsgroups(subset='test', categories=categories, shuffle=True, random
_state=42)
predicted = text_clf.predict(X_test.data)
print(np.mean(predicted == X_test.target))
print(metrics.classification_report(X_test.target, predicted, target_names=X_test.targ
et_names))
print(metrics.confusion_matrix(X_test.target, predicted))
```

Output:

```
0.8348868175765646
                        precision    recall  f1-score   support

           alt.atheism       0.97      0.60      0.74       319
         comp.graphics       0.96      0.89      0.92       389
               sci.med       0.97      0.81      0.88       396
soc.religion.christian       0.65      0.99      0.78       398

              accuracy                           0.83      1502
             macro avg       0.89      0.82      0.83      1502
          weighted avg       0.88      0.83      0.84      1502

[[192   2   6 119]
 [  2 347   4  36]
 [  2  11 322  61]
 [  2   2   1 393]]
```

---

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.

data7_names.csv:

age,sex,cp,trestbps,chol,fbs,restecg,thalach,exang,oldpeak,slope,ca,thal,heartdisease

data7_heart.csv:

```
63.0,1.0,1.0,145.0,233.0,1.0,2.0,150.0,0.0,2.3,3.0,0.0,6.0,0
67.0,1.0,4.0,160.0,286.0,0.0,2.0,108.0,1.0,1.5,2.0,3.0,3.0,2
67.0,1.0,4.0,120.0,229.0,0.0,2.0,129.0,1.0,2.6,2.0,2.0,7.0,1
```

```
import pandas as pd,csv,numpy as np
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
attrs=list(csv.reader(open('data7_names.csv')))[0]
data=pd.read_csv('data7_heart.csv',names=attrs)
print("Head of Data:\n",data.head())
data=data.replace('?',np.nan)
model=BayesianModel([('age','trestbps'),('age','fbs'),('sex','trestbps'),
                     ('exang','trestbps'),('trestbps','heartdisease'),
                     ('fbs','heartdisease'),('heartdisease','restecg'),
                    ('heartdisease','thalach'),('heartdisease','chol')])
model.fit(data,estimator=MaximumLikelihoodEstimator)
data_infer=VariableElimination(model)
print('P(heartdisease|Age=20)')
q=data_infer.query(variables=['heartdisease'],evidence={'age':20})
print(q['heartdisease'])
print('P(heartdisease|cholestrol=100)')
q=data_infer.query(variables=['heartdisease'],evidence={'chol':100})
print(q['heartdisease'])
print('P(heartdisease|trestbps=12)')
q=data_infer.query(variables=['heartdisease'],evidence={'trestbps':12})
print(q['heartdisease'])
```

Output:

Head of Data:

	age	sex	cp	trestbps	chol	...	oldpeak	slope	ca	thal	heartdisease
0	63.0	1.0	1.0	145.0	233.0	...	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	...	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	...	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	...	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	...	1.4	1.0	0.0	3.0	0

[5 rows x 14 columns]
P(heartdisease|Age=20)

heartdisease	phi(heartdisease)
heartdisease_0	0.4931
heartdisease_1	0.2732
heartdisease_2	0.0675
heartdisease_3	0.1338
heartdisease_4	0.0323

P(heartdisease|cholestrol=100)

heartdisease	phi(heartdisease)
heartdisease_0	1.0000
heartdisease_1	0.0000
heartdisease_2	0.0000
heartdisease_3	0.0000
heartdisease_4	0.0000

P(heartdisease|trestbps=12)

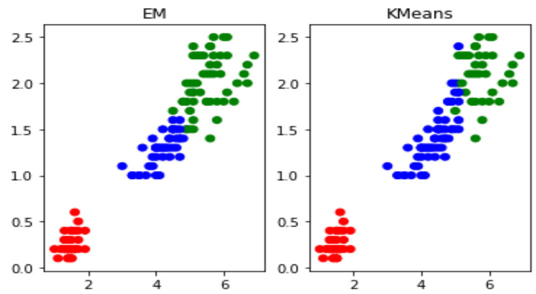
heartdisease	phi(heartdisease)
heartdisease_0	0.1710
heartdisease_1	0.1710
heartdisease_2	0.3160
heartdisease_3	0.1710
heartdisease_4	0.1710

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.

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
np.random.seed(2)
iris=load_iris()
x=pd.DataFrame(iris.data)
y=pd.DataFrame(iris.target)
colormap=np.array(['red','blue','green'])
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=3).fit(x)
plt.subplot(1,2,2)
plt.title("KMeans")
plt.scatter(x[2],x[3],c=colormap[kmeans.labels_])
import sklearn.metrics as sm
print('K Means Accuracy:',sm.accuracy_score(y,kmeans.labels_))
from sklearn.mixture import GaussianMixture
gm=GaussianMixture(n_components=3).fit(x)
ycluster=gm.predict(x)
plt.subplot(1,2,1)
plt.title("EM")
plt.scatter(x[2],x[3],c=colormap[ycluster])
print('EM Accuracy:',sm.accuracy_score(y,ycluster))
print('Confusion Matrix:\n',sm.confusion_matrix(y,ycluster))
```

Output:

```
K Means Accuracy: 0.8933333333333333
EM Accuracy: 0.9666666666666667
Confusion Matrix:
 [[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
```

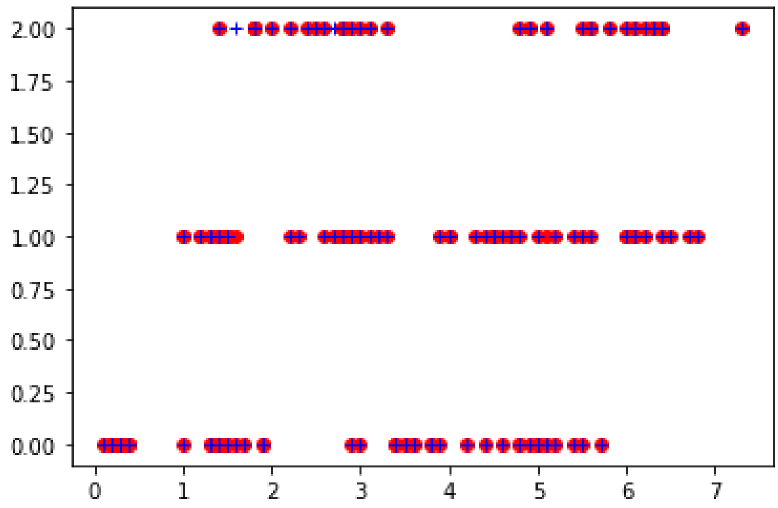


9. Write a program to implement k-Nearest Neighbors algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
iris_dataset=load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset
["target"], random_state=0)
kn = KNeighborsClassifier()
kn.fit(X_train, y_train)
prediction = kn.predict(X_test)
print("ACCURACY:"+ str(kn.score(X_test, y_test)))
plt.plot(X_test,y_test,'ro')
plt.plot(X_test,prediction,'b+')
#target_names = iris_dataset.target_names
#print(target_names)
#for pred,actual in zip(prediction,y_test):
#    print("Prediction:"+str(target_names[pred])+",Actual:"+str(target_names[actual]))
```

Output:

```
ACCURACY:0.9736842105263158
```



---

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.

```
import numpy as np
import matplotlib.pyplot as plt
xtrain=np.array(list(range(3,35))).reshape(32,1)
ytrain=np.sin(xtrain)+xtrain**0.75
xtest=np.array([i/10 for i in range(400)]).reshape(400,1)
ytest=[]
for r in range(len(xtest)):
    w=np.diag(np.exp(-np.sum((xtrain-xtest[r])**2,axis=1)/(2*0.5**2)))
    f1=np.linalg.inv(xtrain.T.dot(w).dot(xtrain))
    params=f1.dot(xtrain.T).dot(w).dot(ytrain)
    pred=xtest[r].dot(params)
```

```
    ytest.append(pred)
plt.plot(xtrain.squeeze(),ytrain,'o')
plt.plot(xtest.squeeze(),ytest,'-')
```

Output: