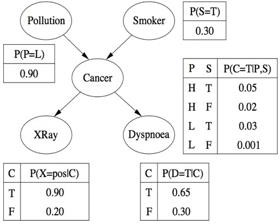
# Program No: 8: BAYESIAN NETWORK

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.

|  |
| --- |
| **Objective**  To write a program to construct a **Bayesian network** considering medical data and to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. |
| **Data Set**  Diagnosis dataset of heart patients |
| **ML Algorithm**  Supervised Learning – **Bayesian Learning** |
| **Description**   * Bayesian networks are very convenient for representing similar probabilistic relationships between multiple events. * Bayesian networks as graphs - People usually represent Bayesian networks as directed graphs in which each node is a hypothesis or a random process. In other words, something that takes at least 2 possible values you can assign probabilities to. For example, there can be a node that represents the state of the dog (barking or not barking at the window), the weather (raining or not raining), etc. * The arrows between nodes represent the conditional probabilities between them — how information about the state of one node changes the probability distribution of another node it’s connected to. |

# Example:



**Program:**

import numpy as np import pandas as pd import csv

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

from pgmpy.inference import VariableElimination #Read the attributes

lines = list(csv.reader(open('P7\_data7\_names.csv', 'r'))); attributes = lines[0]

#attributes = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', # 'oldpeak', 'slope', 'ca', 'thal', 'heartdisease']

#Read Cleveland Heart dicease data

heartDisease = pd.read\_csv('P7\_data7\_heart.csv', names = attributes) heartDisease = heartDisease.replace('?', np.nan)

# Display the data

print('Few examples from the dataset are given below') print(heartDisease.head())

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

# Model Baysian Network

model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),

('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),

('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')]) # Learning CPDs using Maximum Likelihood Estimators print('\nLearning CPDs using Maximum Likelihood Estimators...'); model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network print('\nInferencing with Bayesian Network:') HeartDisease\_infer = VariableElimination(model) # Computing the probability of bronc given smoke.

print('\n1.Probability of HeartDisease given Age=20')

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

print('\n2. Probability of HeartDisease given chol (Cholestoral) =100')

q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={'chol': 100}) print(q['heartdisease'])

# Output:

**Dataset:**

**P7\_data7\_names.csv (14 attributes)** age,sex,cp,trestbps,chol,fbs,restecg,thalach,exang,oldpeak, slope,ca,thal,heartdisease **P7\_data7\_heart.csv (5 instances out of 303)**

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

37.0,1.0,3.0,130.0,250.0,0.0,0.0,187.0,0.0,3.5,3.0,0.0,3.0,0

41.0,0.0,2.0,130.0,204.0,0.0,2.0,172.0,0.0,1.4,1.0,0.0,3.0,0

# Output:

Learing CPDs using Maximum Likelihood Estimators... Inferencing with Bayesian Network:

1. Probability of HeartDisease given Age=28

╒════════════════╤═════════════════════╕

│ heartdisease │ phi(heartdisease) │

╞════════════════╪═════════════════════╡

│ heartdisease\_0 │ 0.6791 │

├────────────────┼─────────────────────┤

│ heartdisease\_1 │ 0.1212 │

├────────────────┼─────────────────────┤

│ heartdisease\_2 │ 0.0810 │

├────────────────┼─────────────────────┤

│ heartdisease\_3 │ 0.0939 │

├────────────────┼─────────────────────┤

│ heartdisease\_4 │ 0.0247 │

╘════════════════╧═════════════════════╛

1. Probability of HeartDisease given chol (Cholestoral) =100

╒════════════════╤═════════════════════╕

│ heartdisease │ phi(heartdisease) │

╞════════════════╪═════════════════════╡

│ heartdisease\_0 │ 0.5400 │

├────────────────┼─────────────────────┤

│ heartdisease\_1 │ 0.1533 │

├────────────────┼─────────────────────┤

│ heartdisease\_2 │ 0.1303 │

├────────────────┼─────────────────────┤

│ heartdisease\_3 │ 0.1259 │

├────────────────┼─────────────────────┤

│ heartdisease\_4 │ 0.0506 │

╘════════════════╧═════════════════════╛

# Program No 9: CLUSTERING BASED ON EM ALGORITHM AND K-MEANS

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same dataset 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.

|  |
| --- |
| **Objective**  **To a**pply **EM algorithm** to cluster a set of data stored in a .CSV file. Use the same data set for clustering using ***k*-Means algorithm** and comparing the results of these two algorithms and  comment on the quality of clustering. |
| **Data Set**  Delivery fleet driver dataset with features “Driver\_id”, “distance\_feature”, “speeding\_feature” having more than 20 instances. |
| **ML Algorithm**  EM algorithm, K means algorithm – Unsupervised clustring |
| **Description**  Expectation Maximization algorithm   * The basic approach and logic of this clustering method is as follows. * Suppose we measure a single continuous variable in a large sample of observations. Further, suppose that the sample consists of two clusters of observations with different means (and perhaps different standard deviations); within each sample, the distribution of values for the continuous variable follows the normal distribution. * The goal of EM clustering is to estimate the means and standard deviations for each cluster so as to maximize the likelihood of the observed data (distribution). * Put another way, the EM algorithm attempts to approximate the observed distributions of values based on mixtures of different distributions in different clusters. The results of EM clustering are different from those computed by k-means clustering. * The latter will assign observations to clusters to maximize the distances between clusters. The EM algorithm does not compute actual assignments of observations to clusters, but classification probabilities. * In other words, each observation belongs to each cluster with a certain probability. Of course, as a final result we can usually review an actual assignment of observations to clusters, based on the (largest) classification probability.   K means Clustering   * The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement. * The algorithm works as follows:   1. First we initialize k points, called means, randomly.   2. We categorize each item to its closest mean and we update the mean’s coordinates, which are the averages of the items categorized in that mean so far.   3. We repeat the process for a given number of iterations and at the end, we have our clusters. * The “points” mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another   method is to initialize the means at random values between the boundaries of the data set |

|  |
| --- |
| (if for a feature x the items have values in [0,3], we will initialize the means with values for x at [0,3]).   * Pseudocode:   1. Initialize k means with random values   2. For a given number of iterations: Iterate through items: |
| Find the mean closest to the item Assign item to mean  Update mean |

# Program:

import matplotlib.pyplot as plt from sklearn import datasets

from sklearn.cluster import KMeans import pandas as pd

import numpy as np

# import some data to play with iris = datasets.load\_iris()

X = pd.DataFrame(iris.data) #print(X)

X.columns = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width'] y = pd.DataFrame(iris.target)

y.columns = ['Targets']

# Build the K Means Model model = KMeans(n\_clusters=3)

model.fit(X) # model.labels\_ : Gives cluster no for which samples belongs to # # Visualise the clustering results

plt.figure(figsize=(14,14))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications using Petal features plt.subplot(2, 2, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40) plt.title('Real Clusters')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

# Plot the Models Classifications plt.subplot(2, 2, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40) plt.title('K-Means Clustering')

plt.xlabel('Petal Length') plt.ylabel('Petal Width') # General EM for GMM

from sklearn import preprocessing

# transform your data such that its distribution will have a # mean value 0 and standard deviation of 1.

scaler = preprocessing.StandardScaler() scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns) from sklearn.mixture import GaussianMixture gmm = GaussianMixture(n\_components=3) gmm.fit(xs)

gmm\_y = gmm.predict(xs) plt.subplot(2, 2, 3)

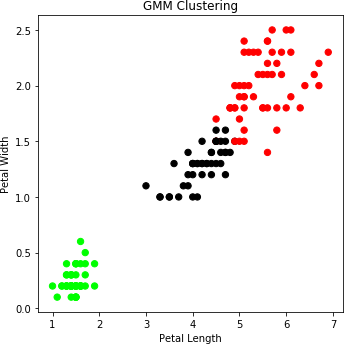
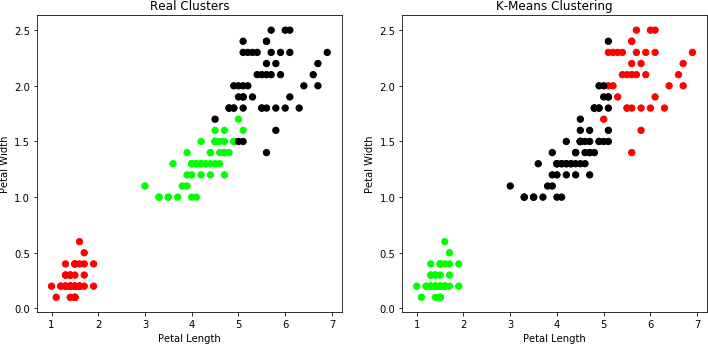
plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[gmm\_y], s=40) plt.title('GMM Clustering')

plt.xlabel('Petal Length') plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')

# Output:

Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.



# Program No 10: K-NEAREST NEIGHBOUR

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.

|  |
| --- |
| **Objective**  To write a program to implement ***k*-Nearest Neighbour algorithm** to classify the iris data set. |
| **Data Set**  IRIS data set with features “petal\_length”, “petal\_width”, “sepal\_lenght” , “sepal\_width” having more than 150 instances. |
| **ML Algorithm**  Supervised Learning - ***k*-Nearest Neighbour algorithm** |
| **Description:**   * K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. * It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data. * Algorithm   Input: Let m be the number of training data samples. Let p be an unknown point. Method:   * 1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).   2. for i=0 to m   Calculate Euclidean distance d(arr[i], p).   * 1. Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.   2. Return the majority label among S. |

# Program:

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...") iris\_data=iris.data iris\_labels=iris.target #print(iris\_data) #print(iris\_labels)

x\_train,x\_test,y\_train,y\_test=train\_test\_split(iris\_data,iris\_labels,test\_size=0.1) print("Dataset is split into training and testing...")

print("Size of trainng data and its label",x\_train.shape,y\_train.shape) print("Size of trainng data and its label",x\_test.shape, y\_test.shape)

# Prints Label no. and their names 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)

# Display the results

print("Results of Classification using K-nn with K=1 ") for r in range(0,len(x\_test)):

print(" Sample:", str(x\_test[r]), " Actual-label:", str(y\_test[r]), " Predicted-label:",str(y\_pred[r])) print("Classification Accuracy :" , classifier.score(x\_test,y\_test))

# Output:

Iris Data set loaded...

Dataset is split into training and testing...

Size of trainng data and its label (135, 4) (135,) Size of trainng 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: [4.8 3. 1.4 0.3] Actual-label: 0 Predicted-label: 0

Sample: [5.8 2.6 4. 1.2] Actual-label: 1 Predicted-label: 1

Sample: [6.7 3.1 4.7 1.5] Actual-label: 1 Predicted-label: 1

Sample: [5.4 3.4 1.5 0.4] Actual-label: 0 Predicted-label: 0

Sample: [5.7 2.5 5. 2. ] Actual-label: 2 Predicted-label: 2

Sample: [6.7 3.3 5.7 2.5] Actual-label: 2 Predicted-label: 2

Sample: [7.7 2.8 6.7 2. ] Actual-label: 2 Predicted-label: 2

Sample: [6.3 2.9 5.6 1.8] Actual-label: 2 Predicted-label: 2

Sample: [6.9 3.1 5.4 2.1] Actual-label: 2 Predicted-label: 2

Sample: [4.7 3.2 1.3 0.2] Actual-label: 0 Predicted-label: 0

Sample: [5. 3.5 1.3 0.3] Actual-label: 0 Predicted-label: 0

Sample: [5.5 4.2 1.4 0.2] Actual-label: 0 Predicted-label: 0

Sample: [5.9 3. 4.2 1.5] Actual-label: 1 Predicted-label: 1

Sample: [6.2 3.4 5.4 2.3] Actual-label: 2 Predicted-label: 2

Sample: [6.1 2.8 4. 1.3] Actual-label: 1 Predicted-label: 1 Classification Accuracy : 1.0

# Program No 11: LOCALLY WEIGHTED REGRESSION ALGORITHM

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

|  |
| --- |
| **Objective**  To implement the non-parametric **Locally Weighted Regression algorithm** in order to fit data points. |
| **Data Set**  The dataset contains billing information based on the attributes total\_bill, tip, sex, smoker, day, time, size. |
| **ML Algorithm**  Instance based learning – **Locally Weighted Regression algorithm** |
| **Description:**   * Given a dataset X, y, we attempt to find a linear model h(x) that minimizes residual sum of squared errors. The solution is given by Normal equations. * Linear model can only fit a straight line, however, it can be empowered by polynomial features to get more powerful models. Still, we have to decide and fix the number and types of features ahead. * Alternate approach is given by locally weighted regression. * Given a dataset X, y, we attempt to find a model h(x) that minimizes residual sum of weighted squared errors. * The weights are given by a kernel function which can be chosen arbitrarily and in my case I chose a Gaussian kernel. * The solution is very similar to Normal equations, we only need to insert diagonal weight matrix W.   Algorithm  def local\_regression(x0, X, Y, tau): # add bias term x0 = np.r\_[1, x0]  X = np.c\_[np.ones(len(X)), X]  # fit model: normal equations with kernel xw = X.T \* radial\_kernel(x0, X, tau)  beta = np.linalg.pinv(xw @ X) @ xw @ Y  # predict value return x0 @ beta  def radial\_kernel(x0, X, tau):  return np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau \* tau)) |

# Program:

import matplotlib.pyplot as plt import pandas as pd

import numpy as np

def kernel(point,xmat, k): m,n = np.shape(xmat)

weights = np.mat(np.eye((m))) # eye - identity matrix 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

def graphPlot(X,ypred):

sortindex = X[:,1].argsort(0) #argsort - index of the smallest xsort = X[sortindex][:,0]

fig = plt.figure()

ax = fig.add\_subplot(1,1,1) ax.scatter(bill,tip, color='green')

ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5) plt.xlabel('Total bill')

plt.ylabel('Tip') plt.show();

# load data points

data = pd.read\_csv('P10\_data10\_tips.csv')

bill = np.array(data.total\_bill) # We use only Bill amount and Tips data tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array mtip = np.mat(tip)

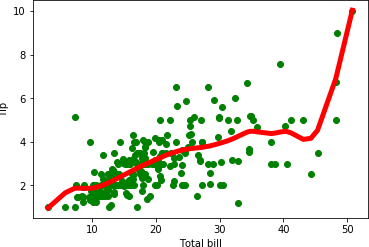
m= np.shape(mbill)[1] one = np.mat(np.ones(m))

X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

ypred = localWeightRegression(X,mtip,2) # increase k to get smooth curves graphPlot(X,ypred)

# Output:

Regression with parameter k = 2



Regression with parameter k = 10

