**Pgm 2**

**import pandas as pd**

**data = pd.read\_csv('C:/Users/navee/Downloads/Book2.csv')**

**concepts = data.iloc[:, 0:-1].values**

**target = data.iloc[:, -1].values**

**def learn(concepts, target):**

**specific\_h = concepts[0].copy()**

**general\_h = [['?' for \_ in range(len(specific\_h))] for \_ in range(len(specific\_h))]**

**for i, h in enumerate(concepts):**

**if target[i] == "Y":**

**specific\_h = [h[x] if h[x] == specific\_h[x] else '?' for x in range(len(specific\_h))]**

**general\_h = [['?' if x != y else specific\_h[x] for x in range(len(specific\_h))] for y in range(len(specific\_h))]**

**if target[i] == "N":**

**general\_h = [[specific\_h[x] if h[x] != specific\_h[x] else '?' for x in range(len(specific\_h))]]**

**general\_h = [h for h in general\_h if h != ['?' for \_ in range(len(specific\_h))]]**

**return specific\_h, general\_h**

**s\_final, g\_final = learn(concepts, target)**

**print("Final specific\_h:")**

**print(s\_final)**

**print("\nFinal general\_h:")**

**print(g\_final)**

**output:**

**specific\_h:**

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

**Final general\_h:**

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

**Pgm 3**

**import numpy as np**

**import matplotlib.pyplot as plt**

**def estimate\_coef(x, y):**

**n = np.size(x)**

**m\_x, m\_y = np.mean(x), np.mean(y)**

**SS\_xy = np.sum(y \* x) - n \* m\_y \* m\_x**

**SS\_xx = np.sum(x \* x) - n \* m\_x \* m\_x**

**b\_1 = SS\_xy / SS\_xx**

**b\_0 = m\_y - b\_1 \* m\_x**

**return b\_0, b\_1**

**def plot\_regression\_line(x, y, b):**

**plt.scatter(x, y, color="m", marker="o", s=30)**

**y\_pred = b[0] + b[1] \* x**

**plt.plot(x, y\_pred, color="g")**

**plt.xlabel('x')**

**plt.ylabel('y')**

**plt.show()**

**def main():**

**x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])**

**y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])**

**b = estimate\_coef(x, y)**

**print("Estimated coefficients: \n b\_0= {} \n b\_1= {}".format(b[0], b[1]))**

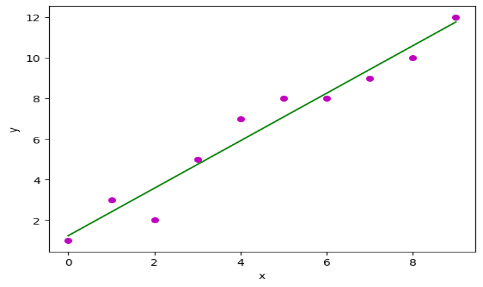
**plot\_regression\_line(x, y, b)**

**main()**

**output**

**Estimated coefficients:**

**b\_0= 1.2363636363636363**

**b\_1= 1.1696969696969697**

**pgm 4**

**import pandas as pd**

**import math**

**def base\_entropy(dataset):**

**p = 0**

**n = 0**

**target = dataset.iloc[:, -1]**

**targets = list(set(target))**

**for i in target:**

**if i == targets[0]:**

**p = p + 1**

**else:**

**n = n + 1**

**if p == 0 or n == 0:**

**return 0**

**elif p == n:**

**return 1**

**else:**

**entropy = 0 - (**

**((p / (p + n)) \* (math.log2(p / (p + n)))**

**+ (n / (p + n)) \* (math.log2(n / (p + n)))))**

**return entropy**

**def entropy(dataset, feature, attribute):**

**p = 0**

**n = 0**

**target = dataset.iloc[:, -1]**

**targets = list(set(target))**

**for i, j in zip(feature, target):**

**if i == attribute and j == targets[0]:**

**p = p + 1**

**elif i == attribute and j == targets[1]:**

**n = n + 1**

**if p == 0 or n == 0:**

**return 0**

**elif p == n:**

**return 1**

**else:**

**entropy = 0 - (**

**((p / (p + n)) \* (math.log2(p / (p + n)))**

**+ (n / (p + n)) \* (math.log2(n / (p + n)))))**

**return entropy**

**def counter(target, attribute, i):**

**p = 0**

**n = 0**

**targets = list(set(target))**

**for j, k in zip(target, attribute):**

**if j == targets[0] and k == i:**

**p = p + 1**

**elif j == targets[1] and k == i:**

**n = n + 1**

**return p, n**

**def Information\_Gain(dataset, feature):**

**Distinct = list(set(feature))**

**Info\_Gain = 0**

**for i in Distinct:**

**Info\_Gain = Info\_Gain + feature.count(i)/ len(feature) \* entropy(dataset, feature, i)**

**Info\_Gain = base\_entropy(dataset) - Info\_Gain**

**return Info\_Gain**

**def generate\_childs(dataset, attribute\_index):**

**distinct = list(dataset.iloc[:, attribute\_index])**

**childs = dict()**

**for i in distinct:**

**childs[i] = counter(dataset.iloc[:, -1], dataset.iloc[:, attribute\_index], i)**

**return childs**

**def modify\_data\_set(dataset,index, feature, impurity):**

**size = len(dataset)**

**subdata = dataset[dataset[feature] == impurity]**

**del (subdata[subdata.columns[index]])**

**return subdata**

**def greatest\_information\_gain(dataset):**

**max = -1**

**attribute\_index = 0**

**size = len(dataset.columns) - 1**

**for i in range(0, size):**

**feature = list(dataset.iloc[:, i])**

**i\_g = Information\_Gain(dataset, feature)**

**if max < i\_g:**

**max = i\_g**

**attribute\_index = i**

**return attribute\_index**

**def construct\_tree(dataset, tree):**

**target = dataset.iloc[:, -1]**

**impure\_childs = []**

**attribute\_index = greatest\_information\_gain(dataset)**

**childs = generate\_childs(dataset, attribute\_index)**

**tree[dataset.columns[attribute\_index]] = childs**

**targets = list(set(dataset.iloc[:, -1]))**

**for k, v in childs.items():**

**if v[0] == 0:**

**tree[k] = targets[1]**

**elif v[1] == 0:**

**tree[k] = targets[0]**

**elif v[0] != 0 or v[1] != 0:**

**impure\_childs.append(k)**

**for i in impure\_childs:**

**sub = modify\_data\_set(dataset,attribute\_index,**

**dataset.columns[attribute\_index], i)**

**tree = construct\_tree(sub, tree)**

**return tree**

**def main():**

**df = pd.read\_csv("C:/Users/navee/Downloads/Book1.csv")**

**tree = dict()**

**result = construct\_tree(df, tree)**

**for key, value in result.items():**

**print(key, " => ", value)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**output:**

**Wind => {'same': (0, 2), 'change': (1, 1)} same => True Humidity => {'warm': (1, 0), 'cool': (0, 1)} warm => False cool => True**

**Pgm 5**

**import numpy as np**

**def sigmoid(x):**

**return 1 / (1 + np.exp(-x))**

**def derivatives\_sigmoid(x):**

**return x \* (1 - x)**

**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**

**epoch = 5000**

**lr = 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))**

**wout = np.random.uniform(size=(hiddenlayer\_neurons, output\_neurons))**

**bout = np.random.uniform(size=(1, output\_neurons))**

**for i in range(epoch):**

**hinp1 = np.dot(X, wh)**

**hinp = hinp1 + bh**

**hlayer\_act = sigmoid(hinp)**

**outinp1 = np.dot(hlayer\_act, wout)**

**outinp = outinp1 + bout**

**output = sigmoid(outinp)**

**EO = y - output**

**outgrad = derivatives\_sigmoid(output)**

**d\_output = EO \* outgrad**

**EH = d\_output.dot(wout.T)**

**hiddengrad = derivatives\_sigmoid(hlayer\_act)**

**d\_hiddenlayer = EH \* hiddengrad**

**wout += hlayer\_act.T.dot(d\_output) \* lr**

**wh += X.T.dot(d\_hiddenlayer) \* lr**

**print("Input:\n", X)**

**print("Actual Output:\n", 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: [[0.89363875] [0.88326031] [0.8928992 ]]**

**Pgm 8:**

**import numpy as np**

**import pandas as pd**

**import csv**

**from pgmpy.estimators import MaximumLikelihoodEstimator**

**from pgmpy.models import BayesianNetwork**

**from pgmpy.inference import VariableElimination**

**lines = list(csv.reader(open("C:/Users/navee/Downloads/p8\_names.csv",'r')));**

**attributes = lines[0]**

**data = pd.read\_csv("C:/Users/navee/Downloads/p8data.csv")**

**data = data.replace('?', np.nan)**

**print('Few examples from the dataset are given below')**

**print(data.iloc[1:6])**

**print('\nAttributes and datatypes')**

**print(data.dtypes)**

**model = BayesianNetwork([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'),**

**('exang', 'trestbps'),('trestbps','heartdisease'),**

**('fbs','heartdisease'), ('heartdisease','restecg'),('heartdisease','thalach'),**

**('heartdisease','chol')])**

**print('\nLearning CPDs using Maximum Likelihood Estimators...');**

**model.fit(data, estimator=MaximumLikelihoodEstimator)**

**print('\nInferencing with Bayesian Network:')**

**HeartDisease\_infer = VariableElimination(model)**

**print('\n1.Probability of HeartDisease given Age=29')**

**q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={'age': 29})**

**print(q)**

**print('\n2. Probability of HeartDisease given chol (Cholestoral) =248')**

**q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={'chol': 248})**

**print(q)**

**pgm 9:**

**import matplotlib.pyplot as plt**

**from sklearn import datasets**

**from sklearn.cluster import KMeans**

**import pandas as pd**

**import numpy as np**

**iris = datasets.load\_iris()**

**X = pd.DataFrame(iris.data)**

**X.columns = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width']**

**y = pd.DataFrame(iris.target)**

**y.columns = ['Targets']**

**model = KMeans(n\_clusters=3)**

**model.fit(X)**

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

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

**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')**

**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')**

**from sklearn import preprocessing**

**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("The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans")**

**pgm 10**

**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**

**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)**

**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 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))**

**pgm 11:**

**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)))**

**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)**

**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();**

**data = pd.read\_csv('C:/Users/navee/Downloads/p11data.csv')**

**bill = np.array(data.total\_bill)**

**tip = np.array(data.tip)**

**mbill = np.mat(bill)**

**mtip = np.mat(tip)**

**m= np.shape(mbill)[1]**

**one = np.mat(np.ones(m))**

**X = np.hstack((one.T,mbill.T))**

**ypred = localWeightRegression(X,mtip,2)**

**graphPlot(X,ypred)**