Prog 5 - NAÏVE BAYESIAN CLASSIFIER

```
# Example of Naive Bayes implemented from Scratch in Python
# Online Resource: https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

import csv
import random
import math

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 = []
	copy = list(dataset)
	while len(trainSet) < trainSize:
		index = random.randrange(len(copy))
		trainSet.append(copy.pop(index))
	return [trainSet, copy]

def separateByClass(dataset):
	separated = {}
	for i in range(len(dataset)):
		vector = dataset[i]
		if (vector[-1] not in separated):
			separated[vector[-1]] = []
		separated[vector[-1]].append(vector)
	return separated

def mean(numbers):
	return sum(numbers)/float(len(numbers))

def stdev(numbers):
	avg = mean(numbers)
	variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
	return math.sqrt(variance)

def summarize(dataset):
	summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
	del summaries[-1]
	return summaries

def summarizeByClass(dataset):
	separated = separateByClass(dataset)
```

```
        summaries = {}
        for classValue, instances in separated.items():
                summaries[classValue] = summarize(instances)
        return summaries

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, inputVector):
        probabilities = {}
        for classValue, classSummaries in summaries.items():
                probabilities[classValue] = 1
                for i in range(len(classSummaries)):
                        mean, stdev = classSummaries[i]
                        x = inputVector[i]
                        probabilities[classValue] *= calculateProbability(x, mean, stdev)
        return probabilities

def predict(summaries, inputVector):
        probabilities = calculateClassProbabilities(summaries, inputVector)
        bestLabel, bestProb = None, -1
        for classValue, probability in probabilities.items():
                if bestLabel is None or probability > bestProb:
                        bestProb = probability
                        bestLabel = classValue
        return bestLabel

def getPredictions(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 (float(correct)/float(len(testSet))) * 100.0


  def main():
        filename = 'pima-indians-diabetes.csv'
        dataset = loadCsv(filename)
        trainingSet=dataset
        testSet=loadCsv('pima-indians-diabetes-test-1.csv')
        print('Records in training data={1} and test data={2} rows'.format(len(dataset),
  len(trainingSet), len(testSet)))
```

```
        # prepare model
        summaries = summarizeByClass(trainingSet)
        # test model
        predictions = getPredictions(summaries, testSet)
        print(predictions)
        accuracy = getAccuracy(testSet, predictions)
        print("Accuracy:",accuracy,"%")

  main()
```

Prog 6 – Naïve Bayes (Doc)

```
import pandas as pd
msg=pd.read_csv('data61.csv',names=['message','label']) #Tabular form data
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('\nThe 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)

# Splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,Y)
print('\nDataset is split into Training and Testing samples')
print('Total training instances :', xtrain.shape[0])
print('Total testing instances :', xtest.shape[0])


# Output of count vectoriser is a sparse matrix
 # CountVectorizer - stands for 'feature extraction'
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain) #Sparse matrix
xtest_dtm = count_vect.transform(xtest)
print('\nTotal features extracted using CountVectorizer:',xtrain_dtm.shape[1])
print('\nFeatures for first 5 training instances are listed below')
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df[0:5])#tabular representation
#print(xtrain_dtm) #Same as above but sparse matrix representation

# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
```

```
print('\nClassstification 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))

#printing accuracy metrics
from sklearn import metrics
print('\nAccuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))

print('Recall :',metrics.recall_score(ytest,predicted),
    '\nPrecison :',metrics.precision_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
```

Prog 7 - BAYESIAN NETWORK

```
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('data7_names.csv', 'r')));
attributes = lines[0]
#attributes = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang',
# 'oldpeak', 'slope', 'ca', 'thal', 'heartdisease']
heartDisease = pd.read_csv('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'])
```

Prog 8 - K-MEANS

```
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)
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.labels_ : Gives cluster no for which samples belongs to
model.fit(X)

# 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')
plt.show()

# 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')
plt.show()

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