Assignment A2

Title: Naive-Bayes Algorithm

Problem statement - Download Pima Indians Diabetes dataset. Use Naive Bayes algorithm for classification
1. Load the data into csv file & split it into training & test dataset
2. Summarize properties in the training dataset so that we ca calculate probability & make predictions.
3. classify samples from the test dataset & a summarized training dataset.

Objective -
- To learn classification algorithms like Naive-Bayes.
- To implement such algorithm to predict data

Outcomes - We will be able to -
- learn classification algorithms
- make predictions using the training dataset.

Theory -
Baye's theorem - It is a way of finding a probability, w we know certain other possibilities.

formula - $P(A/B) = \frac{P(A) \cdot P(B/A)}{P(B)}$

Where, P (A/B) = how often A happens given that B happens
P(B/A) = how often B happens given that A happens.
P(A) how likely A is on its own
P(B) = how likely B is on its own.

Example :- If dangerous fires are rare (1%) but smoke is fairly common (10%) due to barbeques & 90% of dangerous fires make smoke then,

$$P(\text{fire}/\text{smoke}) = \frac{P(\text{fire}) \cdot P(\text{smoke}/\text{fire})}{P(\text{Smoke})}$$

$$= \frac{0.01 \times 0.9}{0.1}$$

$$= 9\%$$

∴ Probability of dangerous fire when there is smoke is 9%

B Naive-Bayes classification

- It is simple, yet effective & commonly used, machine learning classifier. It is probabilistic classifier that makes classificati using the maximum A posteriori decision rule in a Bayesian setting. It can be represented using a very simple bayesian network.
- It is especially popular for text classification & is a traditional solution for problems such as spam detection.

C. Applications -

1. Real time prediction :- Naive Bayes is an eager learning classifier & it is very fast. Thus, it could be used to make prediction in real time.

2. Multi class prediction :- This algorithm is also well known for multiclass prediction feature. Here, we can predict the probability for multiple classes of target variable.

3. Text classification - It is used to have higher success rate as compared to other algorithms. As a result, it is widely

used in spam filtering & sentiment analysis.

Conclusion - Thus, we successfully learnt & implemented Naive - classification algorithm

Test cases:

Input:- Diabetes dataset I

output:-

Confusion matrix

	0	1
0	125	37
1	25	43.

Accuracy = 0.7304.

Test set was 30% of the dataset & 73% of predicted dataset were obtained correctly.

In [51]:
```
import csv
import math
import random


def loadCsv(filename):
    lines = csv.reader(open(r"C:\Users\Viraj Shinde\Desktop\LP1\Pima.csv", "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(*dat
aset)]
    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 x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct/float(len(testSet)))*100.0

def main():
    filename = open(r"C:\Users\Viraj Shinde\Desktop\LP1\Pima.csv")
   # filename = csv.reader(filename)
    #filename = pd.read_csv(r'C:\Users\Viraj Shinde\Desktop\LP1\Pima.csv')
    splitRatio = 0.67
    dataset = loadCsv(filename)
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print(('Split {0} rows into train = {1} and test = {2} rows').format(len(d
ataset),len(trainingSet),len(testSet)))
    #prepare model
    summaries = summarizeByClass(trainingSet)
    #test model
    predictions = getPredictions(summaries, testSet)
    accuracy = getAccuracy(testSet, predictions)
    print(('Accuracy: {0}%').format(accuracy))


main()
```

```
Split 768 rows into train = 514 and test = 254 rows
Accuracy: 73.22834645669292%
```

In [ ]: