1. EXPERIMENT NO: 5

2. TITLE: **NAÏVE BAYESIAN CLASSIFIER**

3. LEARNING OBJECTIVES:

- Make use of Data sets in implementing the machine learning algorithms.
- Implement ML concepts and algorithms in Python

4. AIM:

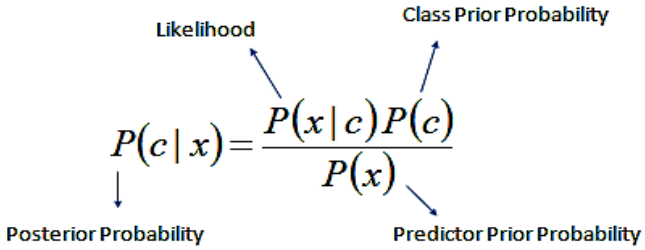
- Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

5. THEORY:

Naive Bayes algorithm : Naive Bayes algorithm is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

where

P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
P(c) is the prior probability of class.
P(x|c) is the likelihood which is the probability of predictor given class.
P(x) is the prior probability of predictor.

The naive Bayes classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function f (x) can take on any value from some finite set V. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values (a1, a2, ... ,an). The learner is asked to predict the target value, or classification, for this new instance.

The Bayesian approach to classifying the new instance is to assign the most probable target value,

v_{MAP}, given the attribute values (a_1, a_2, ..., a_n) that describe the instance.

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n)$$

We can use Bayes theorem to rewrite this expression as

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

$$= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2 \ldots a_n | v_j) P(v_j)$$

Now we could attempt to estimate the two terms in Equation (19) based on the training data. It is easy to estimate each of the P(vj) simply by counting the frequency with which each target value vj occurs in the training data.

The naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent given the target value. In other words, the assumption is that given the target value of the instance, the probability of observing the conjunction a_1, a_2, … , a_n, is just the product of the probabilities for the individual attributes: $P(a_1, a_2, \ldots, a_n \mid v_j) = \Pi_i P(a_i \mid v_j)$. Substituting this, we have the approach used by the naive Bayes classifier.

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$$

where v_{NB} denotes the target value output by the naive Bayes classifier.

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. For example, suppose the training data contains a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class.

Let μ be the mean of the values in x associated with class Ck, and let σ^2_k be the variance of the values in x associated with class Ck. Suppose we have collected some observation value v. Then, the probability distribution of v given a class Ck, p(x=v|Ck) can be computed by plugging v into the equation for a Normal distribution parameterized by μ and σ^2_k. *That is*

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Above method is adopted in our implementation of the program.

Pima Indian diabetis dataset

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

6. PROCEDURE / PROGRAMME :

```
import csv, random, math
import statistics as st

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):
	testSize = int(len(dataset) * splitRatio);
	trainSet = list(dataset);
	testSet =  []
	while len(testSet) < testSize:
	#randomly pick an instance from training data
		index = random.randrange(len(trainSet));
		testSet.append(trainSet.pop(index))
	return [trainSet, testSet]

#Create a dictionary of classes 1 and 0 where the values are the
#instacnes belonging to each class

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

def compute_mean_std(dataset):
	mean_std = [ (st.mean(attribute), st.stdev(attribute))
		for attribute in zip(*dataset)];  #zip(*res) transposes a matrix (2-d array/list)
	del mean_std[-1] # Exclude label
	return mean_std

def summarizeByClass(dataset):
	separated = separateByClass(dataset);
	summary = {} # to store mean and std of +ve and -ve instances
	for classValue, instances in separated.items():
		#summaries is a dictionary of tuples(mean,std) for each class value
		summary[classValue] = compute_mean_std(instances)
	return summary

#For continuous attributes p is estimated using Gaussion distribution
def estimateProbability(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, testVector):
	p = {}
	#class and attribute information as mean and sd
	for classValue, classSummaries in summaries.items():
		p[classValue] = 1
		for i in range(len(classSummaries)):
			mean, stdev = classSummaries[i]
			x = testVector[i] #testvector's first attribute
			#use normal distribution
			p[classValue] *= estimateProbability(x, mean, stdev);
	return p

def predict(summaries, testVector):
	all_p = calculateClassProbabilities(summaries, testVector)
	bestLabel, bestProb = None, -1
	for lbl, p in all_p.items():#assigns that class which has he highest prob
		if bestLabel is None or p > bestProb:
			bestProb = p
			bestLabel = lbl
	return bestLabel

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

dataset = loadCsv('data51.csv');
print('Pima Indian Diabetes Dataset loaded...')
print('Total instances available :',len(dataset))
print('Total attributes present  :',len(dataset[0])-1)

print("First Five instances of dataset:")
for i in range(5):
   print(i+1 , ':' , dataset[i])

splitRatio = 0.2
trainingSet, testSet = splitDataset(dataset, splitRatio)
print('\nDataset is split into training and testing set.')
print('Training examples = {0} \nTesting examples  = {1}'.format(len(trainingSet),
                                                                 len(testSet)))
summaries = summarizeByClass(trainingSet);
predictions = perform_classification(summaries, testSet)

accuracy = getAccuracy(testSet, predictions)
print('\nAccuracy of the Naive Baysian Classifier is :', accuracy)
```

7. RESULTS & CONCLUSIONS:

Sample Result

```
Pima Indian Diabetes Dataset loaded...
Total instances available : 768
Total attributes present  : 8
First Five instances of dataset:
1 : [6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]
2 : [1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0, 0.0]
3 : [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0, 1.0]
4 : [1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0, 0.0]
5 : [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0, 1.0]

Dataset is split into training and testing set.
Training examples = 615
Testing examples  = 153

Accuracy of the Naive Baysian Classifier is : 73.85
```

8. LEARNING OUTCOMES :

- The student will be able to apply naive baysian classifier for the relevent problem and analyse the results.

9. APPLICATION AREAS:

- Real time Prediction: Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- Multi class Prediction: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- Text classification/ Spam Filtering/ Sentiment Analysis: Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- Recommendation System: Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

10. REMARKS: