**PRACTICAL – 9**

**#WAP to implement Native Bayes algorithm**

|  |
| --- |
|  |

from csv import reader

from random import seed

from random import randrange

from math import sqrt

from math import exp

from math import pi

def load\_csv(filename):

dataset = list()

with open(filename, 'r') as file:

csv\_reader = reader(file)

for row in csv\_reader:

if not row:

continue

dataset.append(row)

return dataset

def str\_column\_to\_float(dataset, column):

for row in dataset:

row[column] = float(row[column].strip())

def str\_column\_to\_int(dataset, column):

class\_values = [row[column] for row in dataset]

unique = set(class\_values)

lookup = dict()

for i, value in enumerate(unique):

lookup[value] = i

for row in dataset:

row[column] = lookup[row[column]]

return lookup

def cross\_validation\_split(dataset, n\_folds):

dataset\_split = list()

dataset\_copy = list(dataset)

fold\_size = int(len(dataset) / n\_folds)

for \_ in range(n\_folds):

fold = list()

while len(fold) < fold\_size:

index = randrange(len(dataset\_copy))

fold.append(dataset\_copy.pop(index))

dataset\_split.append(fold)

return dataset\_split

def accuracy\_metric(actual, predicted):

correct = 0

for i in range(len(actual)):

if actual[i] == predicted[i]:

correct += 1

return correct / float(len(actual)) \* 100.0

def evaluate\_algorithm(dataset, algorithm, n\_folds, \*args):

folds = cross\_validation\_split(dataset, n\_folds)

scores = list()

for fold in folds:

train\_set = list(folds)

train\_set.remove(fold)

train\_set = sum(train\_set, [])

test\_set = list()

for row in fold:

row\_copy = list(row)

test\_set.append(row\_copy)

row\_copy[-1] = None

predicted = algorithm(train\_set, test\_set, \*args)

actual = [row[-1] for row in fold]

accuracy = accuracy\_metric(actual, predicted)

scores.append(accuracy)

return scores

def separate\_by\_class(dataset):

separated = dict()

for i in range(len(dataset)):

vector = dataset[i]

class\_value = vector[-1]

if (class\_value not in separated):

separated[class\_value] = list()

separated[class\_value].append(vector)

return separated

def mean(numbers):

return sum(numbers)/float(len(numbers))

def stdev(numbers):

avg = mean(numbers)

variance = sum([(x-avg)\*\*2 for x in numbers]) / float(len(numbers)-1)

return sqrt(variance)

def summarize\_dataset(dataset):

summaries = [(mean(column), stdev(column), len(column)) for column in zip(\*dataset)]

del(summaries[-1])

return summaries

def summarize\_by\_class(dataset):

separated = separate\_by\_class(dataset)

summaries = dict()

for class\_value, rows in separated.items():

summaries[class\_value] = summarize\_dataset(rows)

return summaries

def calculate\_probability(x, mean, stdev):

exponent = exp(-((x-mean)\*\*2 / (2 \* stdev\*\*2 )))

return (1 / (sqrt(2 \* pi) \* stdev)) \* exponent

def calculate\_class\_probabilities(summaries, row):

total\_rows = sum([summaries[label][0][2] for label in summaries])

probabilities = dict()

for class\_value, class\_summaries in summaries.items():

probabilities[class\_value] = summaries[class\_value][0][2]/float(total\_rows)

for i in range(len(class\_summaries)):

mean, stdev, \_ = class\_summaries[i]

probabilities[class\_value] \*= calculate\_probability(row[i], mean, stdev)

return probabilities

def predict(summaries, row):

probabilities = calculate\_class\_probabilities(summaries, row)

best\_label, best\_prob = None, -1

for class\_value, probability in probabilities.items():

if best\_label is None or probability > best\_prob:

best\_prob = probability

best\_label = class\_value

return best\_label

# Naive Bayes Algorithm

def naive\_bayes(train, test):

summarize = summarize\_by\_class(train)

predictions = list()

for row in test:

output = predict(summarize, row)

predictions.append(output)

return(predictions)

# Test Naive Bayes

seed(1)

filename = 'iris.csv'

dataset = load\_csv(filename)

for i in range(len(dataset[0])-1):

str\_column\_to\_float(dataset, i)

# convert class column to integers

str\_column\_to\_int(dataset, len(dataset[0])-1)

n\_folds = 5

scores = evaluate\_algorithm(dataset, naive\_bayes, n\_folds)

print('Scores: %s' % scores)

print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

**OUTPUT:-**

Mean Accuracy % =95.0