Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2021-22

Experiment 4

(Naïve Bayes Classifier)

Aim: Implement Naïve Bayes Classifier on a given Dataset.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:
Dataset 1: Breastcancer.csv
Dataset 2: Social_Network_Ads.csv

1. Perform required preprocessing on Dataset 1 and fit a Naïve Bayes classifier built from scratch. Evaluate the f1 score of classifiers built for categorical and continuous features.
2. Using sklearn library fit a Naïve Bayes classifier on Dataset 2.

```
#using skleran
from sklearn.datasets import load_breast_cancer
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
```

```
data = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(data.data, data.target, test_size=0.3, random_state=42)

gnb = GaussianNB()

gnb.fit(X_train, y_train)

y_pred = gnb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

cr = classification_report(y_test, y_pred)
```

```
import csv
import math
from random import random
```

```
def load_dataset(filename):
    dataset = []
    with open(filename, 'r') as file:
        csv_reader = csv.reader(file)
        next(csv_reader) # skip the first row
        for row in csv_reader:
            # Encode the 'diagnosis' feature as a binary variable
            if row[1] == 'M':
                row[1] = 1.0
            else:
                row[1] = 0.0
            dataset.append([float(x) for x in row])
    return dataset

# Split
def split_dataset(dataset, split_ratio):
    train_size = int(len(dataset) * split_ratio)
    train_set = []
    test_set = list(dataset)
    while len(train_set) < train_size:
        index = int(len(test_set) * random())
        train_set.append(test_set.pop(index))
    return [train_set, test_set]
```

```
def separate_by_class(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        class_value = vector[0]
        if class_value not in separated:
            separated[class_value] = []
        separated[class_value].append(vector)
    return separated

# Calculate the mean of a list of numbers
def mean(numbers):
    return sum(numbers)/float(len(numbers))

#standard deviation
def stdev(numbers):
    if len(numbers) < 1:
        return 0.0
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)


def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del summaries[0]
    return summaries
```

```
def summarize_by_class(dataset):
    separated = separate_by_class(dataset)
    summaries = {}
    for class_value, instances in separated.items():
        summaries[class_value] = summarize_dataset(instances)
    return summaries

# Calculate the Gaussian probability density function for x
def calculate_probability(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

# Calculate the probabilities of predicting each class for a given row
def calculate_class_probabilities(summaries, row):
    total_rows = sum([summaries[label][0][2] for label in summaries])
    probabilities = {}
    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, count = class_summaries[i]
            x = row[i+1]
            probabilities[class_value] *= calculate_probability(x, mean, stdev)
    return probabilities
```

```python
# Predict the class for a given row
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

# Train a Naive Bayes model on a dataset
def train_naive_bayes(train_set):
    summaries = summarize_by_class(train_set)
    return summaries

# Test a Naive Bayes model on a dataset
def test_naive_bayes(summaries, test_set):
    predictions = []
    for i in range(len(test_set)):
        result = predict(summaries, test_set[i])
        predictions.append(result)
    return predictions

# Calculate the accuracy of predictions
def calculate_accuracy(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
```

```
if __name__ == '__main__':

    dataset = load_breast_cancer()


    split_ratio = 0.7
    train_set, test_set = split_dataset(dataset, split_ratio)

    model = train_naive_bayes(train_set)

    predictions = test_naive_bayes(model, test_set)
    actual = [row[0] for row in test_set]
    accuracy = calculate_accuracy(actual, predictions)
    print('Accuracy: {:.2f}%'.format(accuracy))
```