EX NO 9 :   KNN CLASSIFICATION ALGORITHM

Aim:

To implement KNN classification algorithm in python.

Equipments Required:

Hardware – PCs

Anaconda – Python 3.7 Installation / Moodle-Code Runner /Google Colab

Related Theory Concept:

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well –Lazy learning algorithm – KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification. Non-parametric learning algorithm – KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Algorithm:

1. Load the iris CSV file which is downloaded from the Kaggle .

2. Preprocess the data by converting the string column to integer column and then rescale it

3. Determine the min and Max values for each columns and then split the dataset into k folds then calculate the Euclidean distance between two vectors and then print the accuracy of our model.

Program:

/*

Program to implement KNN classification algorithm.

Developed by   : S.SUDHAKARAN

Register Number : 212219040155

*/

k-nearest neighbors on the Iris Flowers Dataset

from random import seed

from random import randrange

from csv import reader

from math import sqrt

```python
# Load a CSV file
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
# Convert string column to float
def str_column_to_float(dataset, column):
        for row in dataset:
            row[column] = float(row[column].strip())
# Convert string column to integer
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
# Find the min and max values for each column
def dataset_minmax(dataset):
        minmax = list()
        for i in range(len(dataset[0])):
                col_values = [row[i] for row in dataset]
                value_min = min(col_values)
                value_max = max(col_values)
                minmax.append([value_min, value_max])
```

```python
        return minmax

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
# Split a dataset into k folds
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
# Calculate accuracy percentage
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
# Evaluate an algorithm using a cross validation split
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)
```

```python
            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
# Calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
        distance = 0.0
        for i in range(len(row1)-1):
            distance += (row1[i] - row2[i])**2
        return sqrt(distance)
# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
        distances = list()
        for train_row in train:
            dist = euclidean_distance(test_row, train_row)
            distances.append((train_row, dist))
        distances.sort(key=lambda tup: tup[1])
        neighbors = list()
        for i in range(num_neighbors):
            neighbors.append(distances[i][0])
        return neighbors
# Make a prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
        neighbors = get_neighbors(train, test_row, num_neighbors)
        output_values = [row[-1] for row in neighbors]
        prediction = max(set(output_values), key=output_values.count)
```

```
        return prediction


# kNN Algorithm
def k_nearest_neighbors(train, test, num_neighbors):
        predictions = list()
        for row in test:
            output = predict_classification(train, row, num_neighbors)
            redictions.append(output)
        return(predictions)
 # Test the kNN on the Iris Flowers dataset
seed(1)
filename = 'data.txt'
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)
# evaluate algorithm
n_folds = 5
num_neighbors = 5
scores = evaluate_algorithm(dataset, k_nearest_neighbors, n_folds, num_neighbors)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

Output:

```
Scores: [96.66666666666667, 96.66666666666667, 100.0, 90.0, 100.0]
Mean Accuracy: 96.667%
```

Result:

Thus the python program successfully implemented KNN classification algorithm.