```
"""CSE 6363 - HWO1 - Part 2
"""
import math
import random

from random import randrange

W = 'W'
M = 'M'

trainData_noAge = {((170, 57), W), ((190, 95), M), ((150, 45), W),
((168, 65), M), ((175, 78), M), ((185, 90), M), ((171, 65), W),
((155, 48), W), ((165, 60), W), ((182, 80), M), ((175, 69), W),
((178, 80), M), ((160, 50), W), ((170, 72), M)}

trainData = {((170, 57, 32), W), ((190, 95, 28), M), ((150, 45, 35), W),
((168, 65, 29), M), ((175, 78, 26), M), ((185, 90, 32), M), ((171, 65, 28), W),
((155, 48, 31), W), ((165, 60, 27), W), ((182, 80, 30), M), ((175, 69, 28), W),
((178, 80, 27), M), ((160, 50, 31), W), ((170, 72, 30), M)}

testX = {(162, 53, 28), (168, 75, 32), (175, 70, 30), (180, 85, 29)}

testX_noAge = {(162, 53), (168, 75), (175, 70), (180, 85)}

class KNN:
    def __init__(self, trainXY, testX, k, precision=4):
        self.precision = precision
        print("--- K: ", k, "  | test datum: ", testX, " ---")
        print()
        print()
        print("training dataset:")
        for xy in trainXY:
            print(xy)
        self.nFold_CrossValidation(trainXY, nfolds=14, k=k, prt=False)
        print()
        print()
        print('test dataset:')
        print(testX)
        self.predict_testset(trainXY, testX, k, precision, prt=True)
        print("--- end of process ---")
        print()
        print()
        print()
        print()
        return

    def predict_testset(self, trainXY, testX, k, precision=4, prt=True):
        self.prt = prt
        self.precision = precision
        self.predictions = list()
        for datum in testX:
            prediction = self.predict_class(trainXY, datum, k)
            self.predictions.append(prediction)
        if self.prt == True:
```

```
        print()
        print("Summary:")
        print('k: ', k)
        print('Test set:', testX)
        print('Predictions:  ', self.predictions)
        print('Precision: ', precision, " sigfig")
    return self.predictions

def euclidean_distance(self, row_A, row_B):
    dist = 0.0
    diffList = list()
    for i in range(len(row_A[0])):
        diff = 0.0
        diff = row_A[0][i]-row_B[i]
        diffList.append(diff)
        dist += (diff)**2
    dist = math.sqrt(dist)
    return round(dist, self.precision)

# Calculate nearest neighbors
def get_neighbors(self, trainX, test_row, k): # trainX, trainX_i, # of nearest neighbors
    distances = list()
    neighbors = list()
    if self.prt == True:
        print()
        print()
        print('Calculate distances to datum: ', test_row)
        print('Training data XY  |   Distance |')
        print('_____________________|_____________|')
    for X_i in trainX:
        dist = self.euclidean_distance(X_i, test_row)
        if self.prt == True:
            print(X_i, ' |  ', dist)
        distances.append((X_i, dist))
    distances.sort(key=lambda tup: tup[1])
    for i in range(k): neighbors.append(distances[i][:])
    #self.print_distances(distances)
    #self.print_neighbors(neighbors)
    if self.prt == True:
        print()
        print(k, ' Nearest Neighbors:')
        for i in neighbors:
            print(i)
    return neighbors

def predict_class(self, trainXY, testX, k):
    #print(trainXY.dtype)
    #print(testX.dtype)
    #self.trainXY = np.asarray([sublist for sublist in trainXY], dtype=object)
    neighbors = self.get_neighbors(trainXY, testX, k)
    output_values = [row[-2][1] for row in neighbors]
    if self.prt == True:
        print()
        print('KNN classes: ', output_values)
```

```
    self.prediction = max(set(output_values), key=output_values.count)
    if self.prt == True:
        print('Datum prediction: ', self.prediction)
    return (self.prediction)

def nFold_CrossValidation(self, trainXY, nfolds, k, prt=False):
    #print("trainXY", trainXY)

    self.scores = list()
    XYfolds = self.split_trainXY(trainXY, nfolds)
    for fold in XYfolds:
        testFold = list()
        trainFolds = list(XYfolds)
        trainFolds.remove(fold)
        trainFolds = sum(trainFolds, [])
        for datum in fold:
            datum_copy = list(datum)
            testFold.append(datum_copy[0])
            datum_copy[-1] = None
            #print("nfold - test datum:", datum_copy)
        prediction = self.predict_testset(trainFolds, testFold, k, prt=False)
        groundtruth = [datum[-1] for datum in fold]
        acc = self.get_acc(groundtruth, prediction)
        self.scores.append(acc)
    print()
    print("Model accuracy is based on training data and ", nfolds, " folds cross validations.")
    print("acc: ", self.scores)
    overall_acc = (sum(self.scores)/float(len(self.scores)))
    overall_acc = round(overall_acc, 2)
    print("Overall model accuracy: ", overall_acc,'%')
    return

# split training dataset into n-folds for cross validation
def split_trainXY(self, trainXY, nfolds):
    trainXY_nfolded = list()
    trainXY_list = list(trainXY)
    fold_size = len(trainXY)/nfolds
    for i in range(nfolds):
        new_fold = list()
        while len(new_fold) < fold_size:
            idx = randrange(len(trainXY_list))
            new_fold.append(trainXY_list.pop(idx))
        trainXY_nfolded.append(new_fold)
    return trainXY_nfolded

def get_acc(self, groundtruth, prediction):
    correct = 0
    for i in range(len(groundtruth)):
        if groundtruth[i] == prediction[i]:
            correct += 1
    return correct / float(len(groundtruth)) * 100.0
```

```
if __name__ == '__main__':

    """ Complete Dataset --- Part a and b """
    KNN(trainData, testX, 1)
    KNN(trainData, testX, 3)
    KNN(trainData, testX, 5)

    """ No Age Dataset --- Part c """
    KNN(trainData_noAge, testX_noAge, 1)
    KNN(trainData_noAge, testX_noAge, 3)
    KNN(trainData_noAge, testX_noAge, 5)
""" end of simple_knn.py """
```