Computer Science 6915 - Winter 2019 Assignment 1

Kaushlender Kumar and Bhumi Tailor

Distance Function: Euclidean distance

We are using the Euclidean distance as a distance function to calculate the distance between two points for our classifier. We decided to use the Euclidean function because of the numerical attributes in the dataset.
The distance between two points in the plane with coordinates (x, y) and (a, b) is given by:

dist((x, y), (a, b)) = sqrt{(x - a)^2 + (y - b)^2}

KNN pseudo-code:

1. **Method to load dataset**

```
def  loadDataSet(trainingFileName, testFileName, trainingDataSet=[], testDataSet=[]):
    with open(trainingFileName,'r') as trainingfile:
        1.1    Read tab delimiter file records
        2.1    Loop through file records and save in array and exclude file header
                a. Convert data point datatype to numerical and save in array
```

2. **Method to calculate distance function**

```
def eculideanDistance(X, Y, length):
        1.1 Loop through clarification data point
            a. Calculate Euclidean distance value between two points
```

3. **Method to calculate neighbors**

```
def filterNeighbors (trainingSet, testInstance, k):
        1.1 Loop through training dataset
            a. Calculate eculideanDistance()

        2.1 Sort all the calculated distance
        3.1 Loop through value of k find out all the instance within the range of k
```

4. **Method to predict the class**

def getResponse(neighbors):

1.1 Loop through all neighbors

a. Calculate instance class

2.1 Sort votes and return predicted class in descending order, in case of even K value it will select highest class number if votes are equal

5. **Method to calculate class probability**

def getProbability(predictedClass):

1.1 Loop through instance neighbors

a. Get neighbors class counter

2.1 Calculate probability of class by dividing, count of number of class / k

3.1 Print the class and probability

Code Validation Test:

We used an excel file to validate the calculated class probability with the result calculated by the program with the different value of K. Additionally, we created a sample code to predict the class and probability using **"sklearn"** python package and then validated the result with the program we created without using any KKN built-in packages.

Command to Execute code: python KNN.py TrainingData_A1.tsv TestData_A1.tsv 4

Reference:

We used the below code reference to adept calculating the neighbour using a separate training and test data set and an added code to estimate the conditional probability, accept command line arguments and handle errors using try..except python block.

https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/
https://www.python-course.eu/k_nearest_neighbor_classifier.php

Code:

```
import csv
import math
import operator
import itertools
import sys
from sys import argv

# function to load training and test data

def loadDataSet(trainingFileName,testFileName,trainingDataSet=[],testDataSet=[]):

    tempTrainingDataSet = []
    tempTestDataSet = []
    num_cols = 0

    try:
        with open(trainingFileName,'r') as trainingfile:
            fileLines= trainingfile.readline()
            lines=csv.reader(trainingfile,delimiter='\t')

            num_cols = len(fileLines.split())

            for row in itertools.islice(lines, 0, None):
                tempTrainingDataSet.append(row)
```

```
        trainingDataset = list(tempTrainingDataSet)

        for x in range(len(trainingDataset)):
            for y in range(num_cols-1):
                trainingDataset[x][y] = float(trainingDataset[x][y])

            trainingDataSet.append(trainingDataset[x])

    with open(testFileName,'r') as testfile:
        fileLines= testfile.readline()
        lines=csv.reader(testfile,delimiter='\t')

        num_cols = len(fileLines.split())

        for row in itertools.islice(lines, 0, None):
            tempTestDataSet.append(row)

        testDataset = list(tempTestDataSet)

        for x in range(len(testDataset)):
            for y in range(num_cols):
                testDataset[x][y] = float(testDataset[x][y])

            testDataSet.append(testDataset[x])
except FileNotFoundError as e:
    print("An Exception Occured : {}".format(e))
except:
```

```
        print("An Exception Occured : {}".format(e.stderr))

# Method to calculate eculidean distance

def eculideanDistance(datapoint1, datapoint2, length):
    pointDistance=0
    for x in range(length):
        pointDistance += pow((datapoint1[x] - datapoint2[x]),2)
    return math.sqrt(pointDistance)

# Method to get neighbors

def filterNeighbors(trainingSet,testInstance,k):
    distances = []
    length = len(testInstance) - 1

    for x in range(len(trainingSet)):
        dist = eculideanDistance(testInstance,trainingSet[x],length)
        distances.append((trainingSet[x],dist))

    distances.sort(key=operator.itemgetter(1))
    neighbors = []

    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors
```

```
def getResponse(neighbors):
    try:
        classVotes = {}

        for x in range(len(neighbors)):
            response = neighbors[x][-1]
            if response in classVotes:
                classVotes[response] += 1
            else:
                classVotes[response] = 1

        sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
    except NameError as e:
        print("An Exception Occured : {}".format(e))
    except:
        print("An Exception Occured : {}".format(e.stderr))

    return sortedVotes[0][0]

# get calculate class probability

def getProbability(predictedClass):

     try:
        totalElement = len(predictedClass)

        if totalElement > 0:
```

```
            print("\n======================Output======================\n")

        for classElement in predictedClass:
            valueCount = predictedClass.count(classElement)
            probability = valueCount/totalElement

            #print('>> predicted=' + repr(classElement) + ', probability=' + repr(probability))
            print(repr(classElement) + '\t' + repr(probability))
    except:
        print("An Exception Occured : {}".format(e.stderr))

def main():

    # read shell argument
    trainingDataSet = []
    testDataSet = []
    kNearest =''

    try:
        if len(argv) >= 3:
            trainingFileName = argv[1]
            testFileName = argv[2]

        if len(argv) == 4:
            kNearest = argv[3]

    except IndexError:
```

```
    print("Excepted training and test file name as an argument.")
except ValueError as e:
    print("An Exception Occured : {}".format(e))
except:
    print("An Exception Occured : {}".format(e.stderr))
else:
    try:
        # prepare data
        loadDataSet(trainingFileName,testFileName,trainingDataSet,testDataSet)

        # generate predictions
        if len(kNearest) <= 0:
            k = 3
        else:
            k = int(kNearest)

        predictedClass = []

        for x in range(len(testDataSet)):
            neighbors = filterNeighbors(trainingDataSet, testDataSet[x], k)
            result = getResponse(neighbors)
            predictedClass.append(int(result))

        #call conditional probability
        getProbability(predictedClass)

    except ValueError as e:
```

```
        print("An Exception Occured : {}".format(e))


if __name__ == "__main__":
    main()
```

Validation code using "sklearn" module:

```
import pandas as pd
import numpy as np

from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)

# Load data set
data = pd.read_csv("TrainingData_A1.tsv",delimiter='\t')
test = pd.read_csv("TestData_A1.tsv",delimiter='\t')
neigh.fit(data.iloc[:,0:9], data['Class'])

# Print predicted class
print(neigh.predict(test))

# Print probability
y_predicted_proba  = neigh.predict_proba(test)
print(y_predicted_proba)
```