K Nearest Neighbor Algorithm Exercise

wangruns

Introduction

In this exercise, you will implements k nearest neighbor algorithm with k-dimension tree and get to see it work on data. Before starting on this programming exercise, we strongly recommend watching the book 《statistical learning method》and understanding what the core concept the algorithm is.

To get started with the exercise, you will need to have some python knowledge, especially numpy and list, but if you don't know python at all, it doesn't matter and you can find instructions on their website to get a quick view.

What is k nearest neighbor

K nearest neighbor is a based algorithm for classification and regression. In this experiment we only implement k nearest neighbor algorithm for classification. Given a new input sample, then find the k samples which are nearest away from the new input sample in the training set, they are defined as k nearest neighbors, finally we will make decision according to the k nearest neighbors, that is to say if most of the k nearest neighbors are apple, so we will classify the new input sample as apple as well. That's the core concept of this algorithm as far as I am concerned.

Now we have already know something about what k nearest neighbor is, the next thing we need to know is:

How to choose the value of k.

How to define the nearest.

How to find the k samples or elements.

In this exercise I will choose k=5, of course you can choose other value as a trial, but if the value of k is too small, the over fitting problem may occur. However if the value of k is too large, the under fitting problem may occur. Actually, you can try a different value to trade off between them. I will use Euclidean distance to define the nearest:

$$L(a,b)=(\sum_{i=1}^{n}|a_i-b_i|^2)^{1/2} \ where \ a=[a_1,a_2,a_3...a_n]^T, b=[b_1,b_2,b_3...b_n]^T$$

You can also choose other formula to define the nearest. As for how find the k samples or elements that are most nearest. It's obvious that we can calculate each distance from the training set to the target sample by linear scanning and then find the most shortest ones as the k nearest

neighbors, which is easy to implement, but may perform slowly especially when the training set is very large. Luckily we can address the problem with the help of k-dimension tree.

Construct k-dimension tree

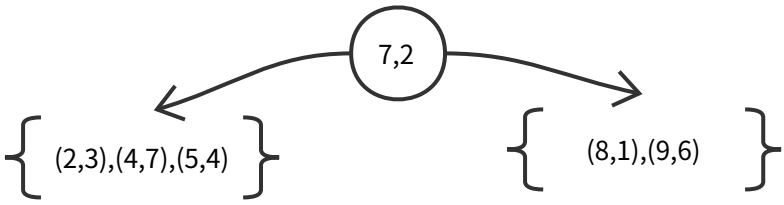
Before we program to implement our k-dimension tree, we need to learn about how k-dimension tree works.

Say that we have a training set of 2 dimension as followed:

$$X=\{(2,3)^T,(5,4)^T,(9,6)^T,(4,7)^T,(8,1)^T,(7,2)^T\}$$

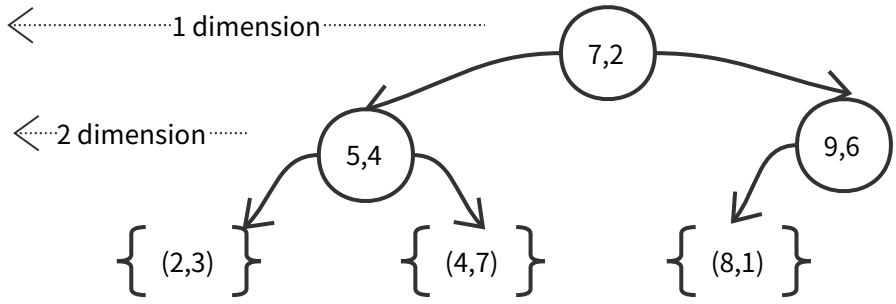
(1) Find the median number among the training set in the 1 dimension space, to be more specific, we can see that {2,5,9,4,8,7} are the numbers in the 1 dimension space, so we can sort them based on the 1 dimension space:

{(2,3),(4,7),(5,4),(7,2),(8,1),(9,6)}, and (7,2) is the median one, choose it as the root node of the k-dimension tree:



(2) Find the median number among the training set in the 2 dimension space, to be more specific, we can see that {3,7,4} are the numbers in the 2 dimension space on the left sub tree, and {1,6} are the numbers in the 2 dimension space on the right sub tree. Actually, what we need to is to repeat the steps in the (1) recursively except for the change of dimension. On the left sub tree, we sort it based on the 2 dimension space:

{(2,3),(5,4),(4,7)}, and (5,4) is the median one, choose it as the root node of the k-dimension tree on this layer. In the same way we can get that (9,6) is the median one on the right sub tree, and choose it as root node of the k-dimension tree on this layer as well:



Naturally, if we want to use k-dimension tree, the first we need to do is to construct this data structure to store our training set samples whatever language we use for programming. Here I will use Python to implement it.

Classification exercise

There are 150 instances in the training set and test set totally. Each instance has 4 attributes(sepal length in cm, sepal width in cm, petal length in cm, petal width in cm). The class are Iris Setosa, Iris Versicolour and Iris Virginica. The training set looks like as followed:

4.7,3.2,1.3,0.2,Iris-setosa

4.6,3.1,1.5,0.2,Iris-setosa

……

and the test set looks like as followed:

5.1,3.5,1.4,0.2

4.9,3.0,1.4,0.2

……

Our work is to get our classifier via the training set and use it to make decision on test set. Actually, we can use 3 steps to address the problem.

Part 1: Data preprocessing

Load the data and transform the form of data to what we need i.e vector or matrix.

```
## ====================Part 1: Data Preprocessing====================
    #  Load the data and transform the form of data to what we need. i.e
    #  vector or matrix
    #training set and test set input
    with open('./training.txt') as f:
        for line in f:
            row=line.rstrip().split(',');
            row[4]=labels[row[4]];
            global trainSet;
            trainSet.append(row);
    trainSet=np.array(trainSet).astype(float);
    with open('./test.txt') as f:
        for line in f:
            row=line.rstrip().split(',');
```

```
            testSet.append(row);
    Xtest=np.array(testSet).astype(float);
```

Part 2: Construct kd tree

Construct the k nearest neighbor tree with our training set.

```
## ====================Part 2: Construct KD Tree====================
    #  Construct the k nearest neighbor tree with our training set
    #
    #Construct KD Tree
    kdTree=constructKDTree(trainSet,0);
```

Part 3: Prediction by knn

Predict the result via k nearest neighbors algorithm, which will fist find k nearest neighbors via our kdTree and then make decision via majority voting rule.

```
## ====================Part 3: Prediction By knn====================
    #   Predict the result via k nearest neighbors algorithm
    #   which will fist find k nearest neighbors via our kdTree and then
    #   make decision via majority voting rule
    global K;
    K_backup=K;
    for xtest in Xtest:
        neighbors=list();
        findKNN(kdTree,xtest,0,neighbors);
        K=K_backup;
        hypothesis=majorityVoting(neighbors);
        print xtest,hypothesis;
```

Programming codes

```
#coding:utf8
import sys
import numpy as np


answer = []
```

```
trainSet = []
testSet = []
#K is the number of nearest neighbors
K = 5
#labels of the samples in the training set
labels={'Iris-setosa':'0','Iris-versicolor':'1','Iris-virginica':'2'};

class Node:
    def _init_(self,val=[],left=None,right=None):
        self.val=val;
        self.left=left;
        self.right=right;

def constructKDTree(X,l):
    if X.size==0:
        return;
    root=Node();
    dimension=l%5 if l!=4 else (l+1)%5;
    column=X[:,dimension];
    X=X[column.argsort()];
    middle=column.size/2;
    root.val=X[middle];
    root.left=constructKDTree(X[:middle],l+1);
    root.right=constructKDTree(X[middle+1:],l+1);
    return root;

def updateCurrentNearestNode(curNode,target,neighbors):
    curDistance=sum((curNode[:-1]-target)**2)**0.5;
    global K;
    if K>0:
        neighbors.append([curDistance,curNode[-1]]);
```

```
            K-=1;
        else:
            neighbors.sort();
            if curDistance<neighbors[-1][0]:
                neighbors[-1]=[curDistance,curNode[-1]];

def findKNN(kdTree,target,l,neighbors):
    if kdTree is None:
        return;
    dimension=l%5 if l!=4 else (l+1)%5;
    value=target[dimension];
    nextStepIsLeft=False;
    if value<kdTree.val[dimension]:
        findKNN(kdTree.left,target,l+1,neighbors);
        nextStepIsLeft=True;
    else:
        findKNN(kdTree.right,target,l+1,neighbors);
    updateCurrentNearestNode(kdTree.val,target,neighbors);
    brotherArea=kdTree.right if nextStepIsLeft else kdTree.left;
    if brotherArea is not None:
        d=abs(value-kdTree.val[dimension]);
        neighbors.sort();
        if d < neighbors[-1][0]:
            findKNN(brotherArea,target,l+1,neighbors);

def majorityVoting(neighbors):
    dic=dict();
    for nei in neighbors:
        dic[nei[1]]=dic.get(nei[1],0)+1;
    maxV=None;classK=None;
    for k,v in dic.items():
```

```
            if maxV is None or v>maxV:
                classK=k;maxV=v;
        return classK;


def classify():
        ## ====================Part 1: Data Preprocessing====================
        #  Load the data and transform the form of data to what we need. i.e
        #  vector or matrix
        #training set and test set input
        with open('./training.txt') as f:
            for line in f:
                row=line.rstrip().split(',');
                row[4]=labels[row[4]];
                global trainSet;
                trainSet.append(row);
        trainSet=np.array(trainSet).astype(float);
        with open('./test.txt') as f:
            for line in f:
                row=line.rstrip().split(',');
                testSet.append(row);
        Xtest=np.array(testSet).astype(float);



        ## ====================Part 2: Construct KD Tree====================
        #  Construct the k nearest neighbor tree with our training set
        #
        #Construct KD Tree
        kdTree=constructKDTree(trainSet,0);



        ## ====================Part 3: Prediction By knn====================
```

```
        #   Predict the result via k nearest neighbors algorithm
        #   which will fist find k nearest neighbors via our kdTree and then
        #   make decision via majority voting rule
        global K;
        K_backup=K;
        for xtest in Xtest:
                neighbors=list();
                findKNN(kdTree,xtest,0,neighbors);
                K=K_backup;
                hypothesis=majorityVoting(neighbors);
                print xtest,hypothesis;


if __name__ == '__main__':
    classify()
```