Machine Learning

Program assignment #2

0410001

Hong-Shuo, Chen

1. Abstract

Implement Kd-Tree and use K-NN classifier to analyze a data set.

1. Problem

* Construct K-NN classifier with Kd-Tree and use Kd-Tree to find the designated nearest instance
* Distance metric: Euclidean distance

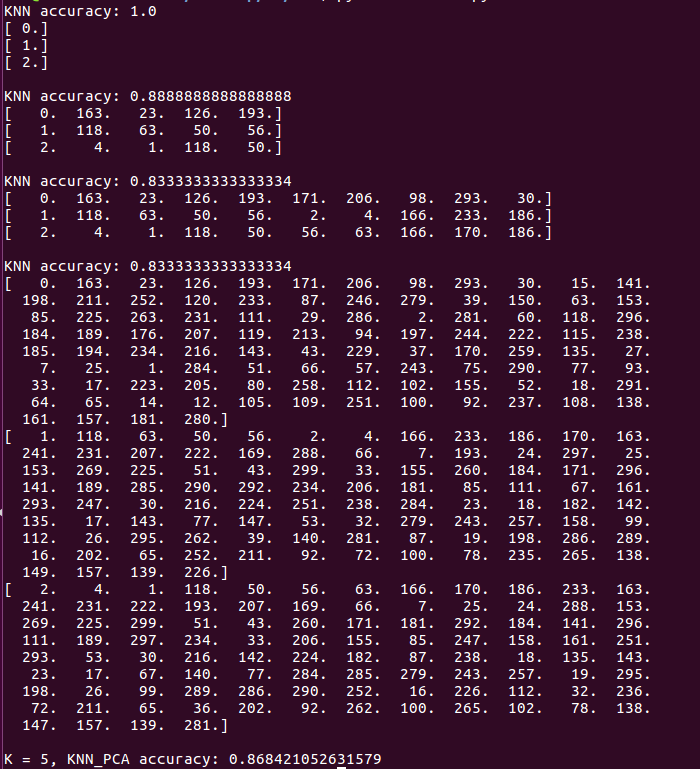
1. Environment

* Ubuntu 16.04.3 LTS

1. Using library and language

* Library: numpy, math, sys, copy
* Language: Python 3.5.2

1. Results



1. Algorithm

I define the tree node as below as a container for my KD-tree. There are four elements in each node, which respectively are parent, left, right, and data. We store our information and the attribute in the data.

For example,

root = Tree( ),

root.data = [data[0], attribute],

while data[0] is our information and the attribute is the attribute we use to classify the left node and the right node.

class Tree(object):

def \_\_init\_\_(self):

self.parent = None

self.left = None

self.right = None

self.data = None

There are three important parts, respectively which are creating the KD-tree, finding the nearest node, and KNN. I will explain them in detail below.

* Create KD-tree

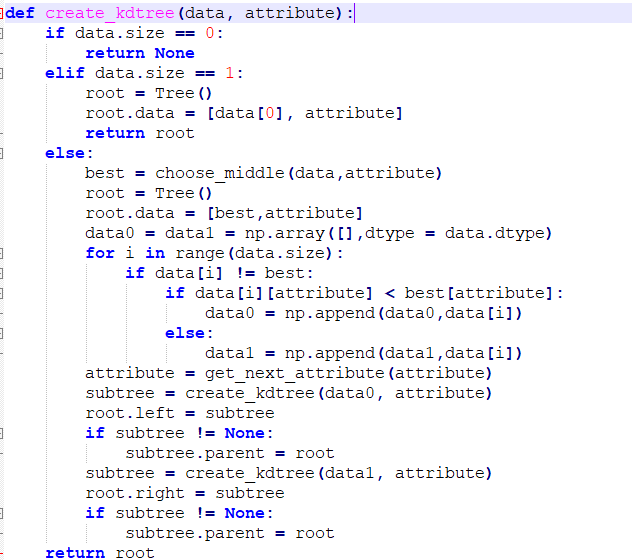
We need two parameters in this function, which respectively are data and attribute. I build the tree with recursive method.

There are three conditions.

When the dataset is empty, we return None.

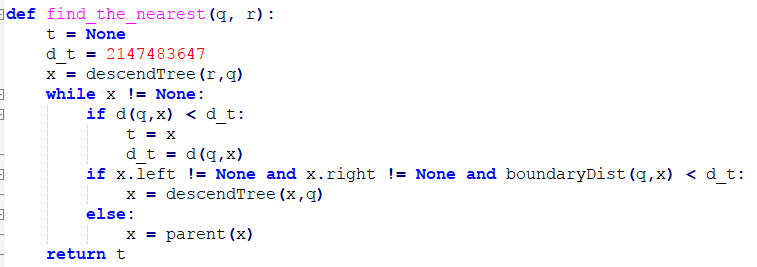
While the size of the dataset is 1, we build a tree node which the data is [data[0], attribute] and return it.

In the last condition, the size of the data is bigger than 1, and we need to split the data with the attribute we get from the parameter. We find the middle of the dataset, and build the tree, which the data is [best, attribute]. We split the remaining dataset into two parts. The first is smaller than the middle node, and the second part is bigger than the middle node. Finally, we call this function again, set the left of the root is create\_kdtree(data0, attribute) and set the right of the root is create\_kdtree(data1, attribute), and return the root.



* Find the nearest node

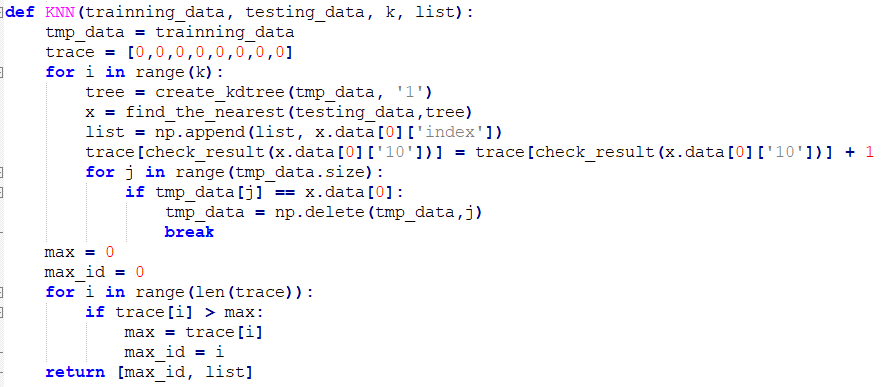
In this function, we get the query and the root node of the tree. t is our target node, while d\_t is the distance between the query and the target node. While x is not None, which means the x is not the parent of the root of the tree, so we need to keep finding. If the distance between the q and the x is less than d\_t, we need to update t and d\_t. If x is not the leaf node and the boundaryDist(q,x) is less than d\_t, we search the subtree, so we go down the subtree to the leaf node, else we go to our parent and prune the subtree. In the end, it will return the nearest node.



* KNN

In this function, there are four parameters, which are training data, testing data, k, and the list. We could set the k as we want, such as 1, 5, 10, 100...and so on. The list is used to keep tack to the k nearest data.

Trace is used to know which class appears most. I create the tree and find the first nearest data, and then I delete that data from the dataset, and create the tree and find the nearest data again until we find the k nearest data. Finally, we return we return the predicted result and the list.



In order to complete the three primary function above, I define the following function below:

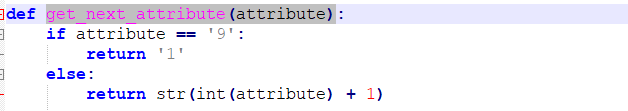
* get\_next\_attribute(attribute)

get the next attribute

Parameters:

* + attribute: str

return: attribute: str



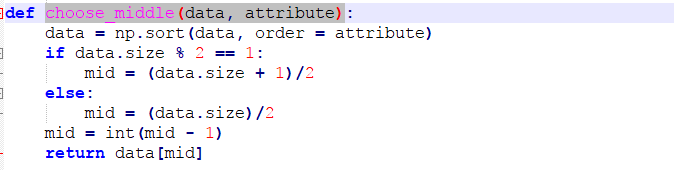
* choose\_middle(data, attribute)

choose the middle data In the dataset in the order of the attribute

Parameters:

* + data: numpy.ndarray
  + attribute: str

return: data[mid]: numpy.ndarray



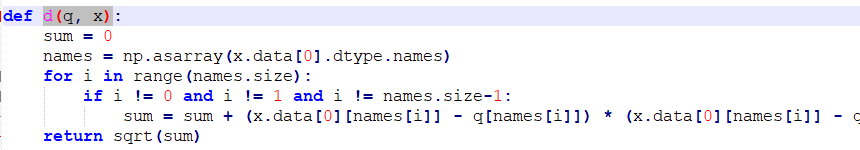
* d(q, x)

calculate the distance between the query and the given node

Parameters:

* + q: numpy.ndarray.dtype
  + x: tree node

return: distance: float64



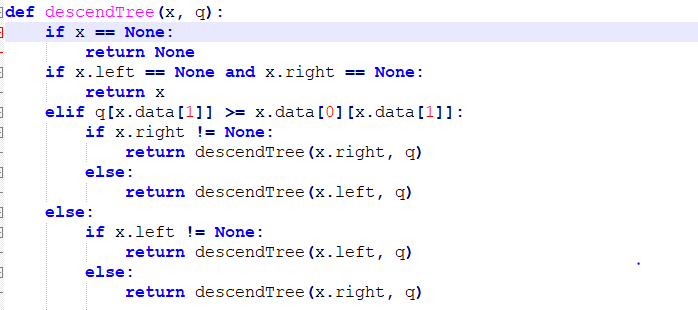
* descendTree(x, q)

go to the leaf node

Parameters:

* + q: numpy.ndarray.dtype
  + x: tree node

return: leaf node: tree node



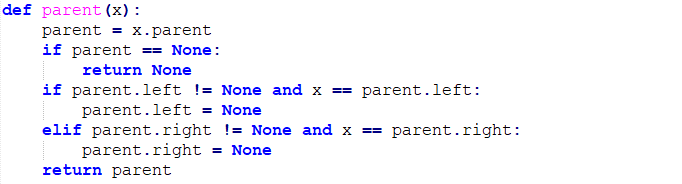
* parent(x)

return the parent of x and prune the tree

Parameters:

* + x: tree node

return: parent: tree node



* boundaryDist(q, x)

return distance between q and boundary

Parameters:

* + q: numpy.ndarray.dtype
  + x: tree node

return: distance: float64



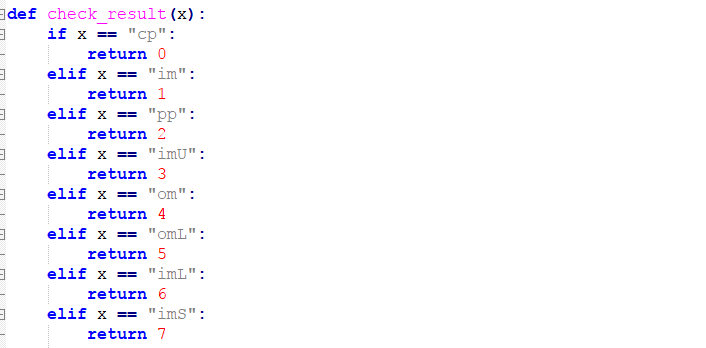
* check\_result(x)

return the id of the result

Parameters:

* + x: str

return: id: int



* main(k)

do all the things

Parameters:

* + k: int

return:

1. Bonus: PCA

In the function, we do the following things. Firstly, we get the A matrix which we need to calculate the average of the data and subtract the original data from the average. We use the package of the numpy to calculate the eigenvalue(w) and the eigenvector(v) of the transpose of A dot A. I set the threshold to the 0.98, so I need to select the 7 biggest eigenvalue, and the dimension of the data will decrease from 9 to 7. After get the Matrix Q, we need to process our testing data as well. Finally, we put the data which is processed into our function defined above, and it will get the similar result. My accuracy rate is about 0.868, it is a little worse than the accuracy without using PCA which is 0.888. However, due to the dimension is lower, I think this result is predictable. We need less time to process the data, but the trade-off is the accuracy rate.

