**Machine Learning Homework 2**

**0416206 李明峻**

**Environment**: Ubuntu14.04 LTS

**Language**: c++

**Using Library**:

#include <iostream>

#include <string>

#include <fstream>

#include <vector>

#include <stdio.h>

#include <stdlib.h>

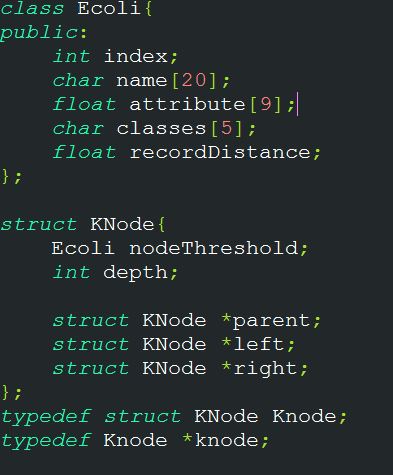
#include <string.h>

#include <iomanip>

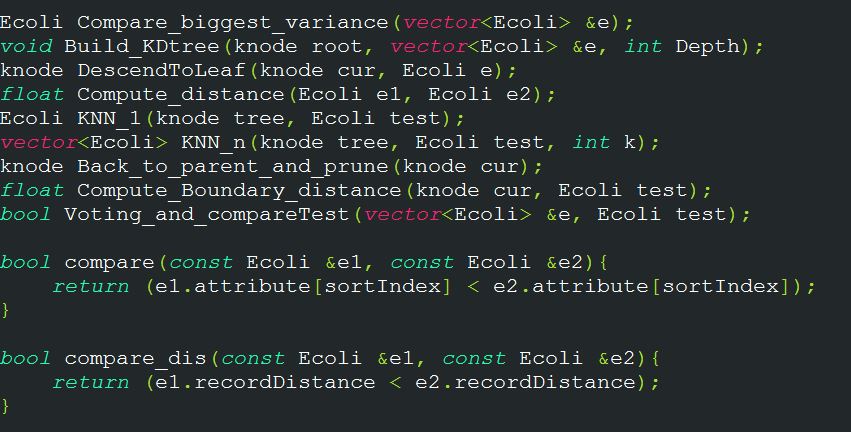
#include <algorithm>

#include <math.h>

**Explanation of code**:

**class and struct declaration**:****

1. **Ecoli :** store Ecoli’s information read from ‘train.csv’ , including index and Ecoli’s name and 9 attribute values and class name , the last variable recordDistance is to record the distance between testdata and the currently found node(Ecoli) .
2. **Knode:** nodeThreshold specify the node Ecoli’s information , if the node is non-leaf node , then nodeThreshold store the median Ecoli sorted by the specified attribute ; else if the node is leaf node , then nodeThreshold store just Ecoli information . so each node in the tree will have only one Ecoli stored in it , also store its parent and left right child node pointer .

****

**Function declaration:**

**1.** **Compare\_biggest\_variance() :**

This function will compute 9 attributes’ variance and select the biggest variance dimension , then sort all the data by this dimension and select the median data and return it .

**2. Build\_KDtree() :**

This is the function building KD-tree mainly , in this function’s argument , we will send Ecoli information which are going to be classified , and throw this information into Compare\_biggest\_variance() , then the return value will be stored in nodeThreshold of this node , comparing each data by specified attribute , and the smaller data traverse to left child and bigger data traverse to right child , before this , we need to build left and right child node , and build all the tree by recursively calling this function until the classified data left only one(leaf node) .

**3.DescendToLeaf():**

This function is simply let the input node traverse to the leaf node of this KD-tree by the rules when building KD-tree .

**4. Compute\_distance():**

This function will compute the Euclidean distance between two Ecolis .

**5. KNN\_1():**

This function implement KNN when K = 1 : first step we traverse the tree form its root , and go down to leaf node by the rules when building KD-tree , and every time when the node traverse to leaf node , we will compute the distance between testdata and current node , if the distance is shorter than we have ever found , then update the distance and neighbor node information , after checking this , let the node go back to its parent and prune this path out , and every time the node go to non-leaf node , we will check the distance between testdata and current node by the specified attribute(dimension) , and compare the currently minimum distance , if smaller , it represents there’s possibly exist shorter distance on the other side of this nodeThreshold’s dimension , so we have to traverse down to leaf node again and find whether if there’s exist shorter distance , repeatedly doing this until the node go back to root node of this tree , eventually , we will get the shortest distance , return the nearest neighbor .

**6. KNN\_n():**

This function implement KNN with K > 1 , same way as K = 1 , the different is that we create a vector to store nearest K node , first we push K Ecoli data into this vector when traversing the tree , after collecting K data , we traverse the same way as before , and keep updating the distance when finding shorter distance .

**7. Back\_to\_parent\_and\_prune():**

Let the current node go back to parent and prune this path out .

**8. Compute\_Boundary\_distance():**

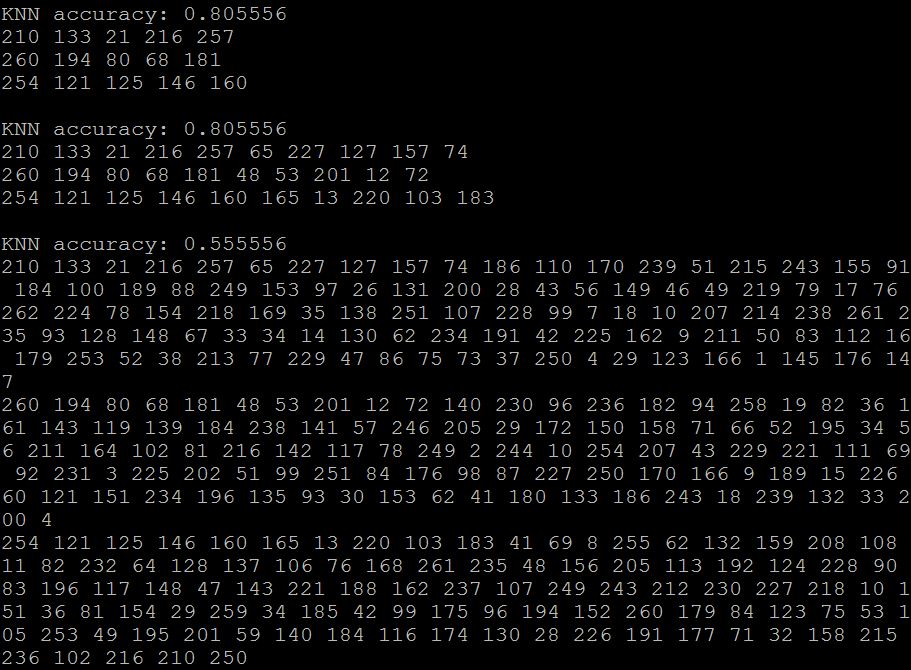
When the node is non-leaf node , we will compute its distance between testdata and current node by its selected attribute(dimension) .

**9. Voting\_and\_compareTest():**

In this function , implement voting and after select the winning candidate , compare it to the original testdata , if same then return true , otherwise return false .

**\*note: I take the 36 testdata from ‘train.csv’ , and other 264 data is for training(build KD-tree).**

**Result:**

****