Indian Institute of Technology, Kharagpur
Department of Computer Science and Engineering
Software Engineering (CS 20202), Spring 2024

Assignment 5 – C++ Programming

Total marks: 100

Grading guidelines:

1. *Zero marks for a submission if it does not pass the plagiarism test.*
2. *Break-up of Credits will be as follows:*
   (a) *Percentage of features implemented: 70%*
   (b) *Code understanding – code clarity, comments: 10%*
   (c) *Whether reasonably able to answer questions: 20%*

In the previous assignment, you have implemented the following abstract data type (ADT) called *DataVector*:

```
class DataVector {
    vector<double> v;
    public:
    DataVector(int dimension=0);
    ~DataVector();
    DataVector(const DataVector& other);
    DataVector & operator=(const DataVector &other);
    void setDimension(int dimension=0);
    DataVector operator+(const DataVector &other);
    DataVector operator-(const DataVector &other);
    double operator*(const DataVector &other);
}
```

Using the above ADT, you have implemented a simple approximate nearest neighbour search (ANN) algorithm which given a test vector *v* and a vector dataset *D*, quickly find other vectors *v' in D* which are closest to *v*. You have run and tested your algorithm on the Fashion MNIST dataset from the following link:
https://github.com/zalandoresearch/fashion-mnist

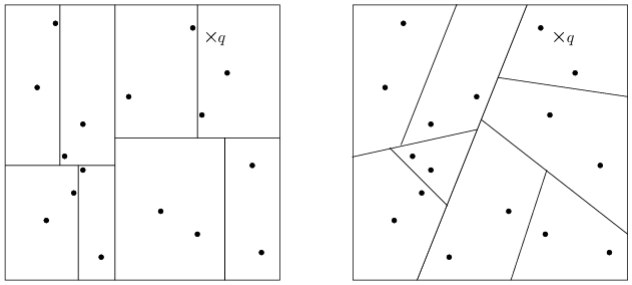
In this assignment, you have to implement a class of **ANN indices**, which are tree-based indices. An **ANN index** is a data structure that stores a dataset of vectors in a format that can be used to quickly search for the k-nearest neighbors of a given test vector. A tree-based index is a one where, the set of vectors are arranged in a binary tree hierarchy. We start with the full dataset at the root node, and at each node n the set S_n is split in two parts using a rule, and the children of the node n represent the two splits of S_n. The rules for splitting are designed such that vectors in a node represent a spatial region in the vector space. For any tree-based index, given a test point, we can reach a leaf node which contains the region to which the test point must reach. The search algorithm should backtrack the recursion path to the root node. At each node, it should search all the points in the region corresponding to the parent node, if the current farthest point from the query point (in the k-NN list) is nearer than the distance of the query point from

the boundary with the sibling. The distance from the boundary can be calculated easily since the splitting hyperplane, which is part of the rule is stored at each node. You have to implement two algorithms from the following paper:

- Dasgupta, Sanjoy, and Yoav Freund. "Random projection trees and low dimensional manifolds." In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 537-546. 2008.

Available at Link: https://cseweb.ucsd.edu/~dasgupta/papers/rptree-stoc.pdf

The algorithms are: kd-tree (left in figure) and RP-tree (right in figure). The following figure show the regions for both these trees.



The following are the different algorithms from the paper:

procedure MAKETREE(S)
if $|S| < MinSize$ **return** $(Leaf)$
$Rule \leftarrow$ CHOOSERULE(S)
$LeftTree \leftarrow$ MAKETREE($\{x \in S : Rule(x) = \text{true}\}$)
$RightTree \leftarrow$ MAKETREE($\{x \in S : Rule(x) = \text{false}\}$)
return $([Rule, LeftTree, RightTree])$

The k-d tree CHOOSERULE picks a coordinate direction (typically the coordinate with largest spread) and then splits the data on its median value for that coordinate.

procedure CHOOSERULE(S)
comment: k-d tree version
choose a coordinate direction i
$Rule(x) := x_i \leq \text{median}(\{z_i : z \in S\})$
return $(Rule)$

procedure CHOOSERULE(S)
comment: RPTree-Max version
choose a random unit direction $v \in \mathbb{R}^D$
pick any $x \in S$; let $y \in S$ be the farthest point from it
choose δ uniformly at random in $[-1, 1] \cdot 6\|x - y\|/\sqrt{D}$
$Rule(x) := x \cdot v \leq (\text{median}(\{z \cdot v : z \in S\}) + \delta)$
return $(Rule)$

Here, S is the set of vectors at the current node. D is the dimension of the vector.

The base class `TreeIndex` defines the basic functionalities of an index. Since an index is a large data structure, there should be one copy of it which should store all the data to be searched. Hence, the class `TreeIndex` should be a singleton class. There should be two derive classes `KDTreeIndex` and `RPTreeIndex`.

Further, any concrete ANN index should have the following properties / function:

- **GetInstance:** a static method which generates a new instance of an index or return the existing instance. Also, implement the constructors. **[10 marks]**
- **AddData / RemoveData:** add or remove data from the `VectorDataset` in the current index. **[10 marks]**
- **MakeTree:** create the tree data structure, which is used to the store the vectors in a hierarchy. Implement the **ChooseRule** function. **[30 marks]**
- **Search:** given a test point, find the k-nearest neighbors. **[20 marks]**

Write your code in a header file TreeIndex.h and functions in source code file TreeIndex.cpp. Submit both the files to moodle.

You can use the following code structure and add the necessary functions:

```
class TreeIndex {
protected:
  TreeIndex() {}
  ~TreeIndex() {}

public:
  static TreeIndex& GetInstance();
};

class KDTreeIndex : public TreeIndex {
public:
  static KDTreeIndex& GetInstance();
private:
  KDTreeIndex() {}
  ~KDTreeIndex() {}
};


class RPTreeIndex : public TreeIndex {
public:
  static RPTreeIndex& GetInstance();
private:
  RPTreeIndex() {}
  ~RPTreeIndex() {}
};
```