Description for classes and files

- Classes
  - CSVTable.hpp : data table / loads .csv file into the table
  - KdNode.hpp: node / train phase is implemented here / data are split into the nodes / splitting axis and values are determined.
  - KdTree.hpp: tree / test phase is implemented here / 1nn search is done by traversing the tree / also saves and loads tree.
  - QueryTable.hpp: query result table / saves the query result.

- Applications
  - Build_kdtree.cpp
  - Query_kdtree.cpp

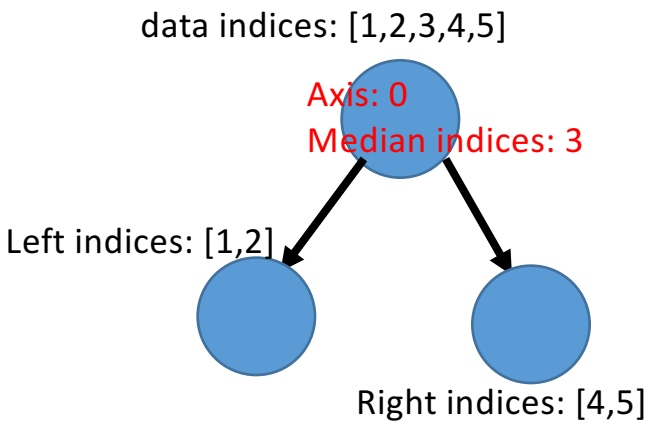
Implementation details

- ./build_kdtree:
  - Takes two arguments: the train data path and path to save the model.
  - Optionally, the existing sample_data.csv can be loaded.
- ./query_kdtree:
  - Takes four arguments: train data path, model path, test data path, and path to save the query result.
  - Optionally, the existing sample_data.csv and precomputed model can be loaded.

- CSVTable.hpp:
  - The data is stored in a class CSVTable, e.g. CSVTable data.
  - The data points are accessed via indexing, e.g. data[i].
  - In entire implementation, instead of using the data value itself, the index (i,j) are kept to represent the data.

Implementation details for KdNode.hpp

e.g. lets say we have Data = [3 5 7 9 11].
(Here we have only 1 axis, so splitting axis is always 0, 0 based).
Median value is 7, and its indices is 3.



Training Phase:

1. The indices for train data is assigned to root.
2. The splitting axis is determined by either one of:
   a. Maximizing Variance
   b. Minimizing Skew
   c. Minimizing Kurtosis
3. The median point /wrt splitting axis is selected.
4. The median point becomes the root's point.
5. Remaining train data is split into left and right child:
   a. If smaller than median point -> to left
   b. If larger than median point -> to right
6. This is repeated for all child nodes, until the leaf if reached.

Implementation Details for KdTree.hpp

Testing Phase:

1. At node n; find the distance between the query point and the point representing the node (which always one of the train point).
2. If distance is smaller than previous distance, update the new smallest distance, and the data point.
3. Traverse down the tree:
   a. If distance to the hyperplane < than the bound: go both
   b. If query point is left side of the hyperplane: go left
   c. If query point is right side of the hyperplane: go right
4. Repeat it until the leaf is reached

Runtime choices

- Algorithms to determine the Splitting axis:
  a. Maximizing Variance
  b. Minimizing Skew
  c. Minimizing Kurtosis

- Bound to the splitting hyperplane:
  - Default is set to 0.1.
  - Bound to the splitting hyperplane is sensitive to the data, and thus left to be chosen at the runtime.
    - If too large, then computation complexity increases because there are higher chances of traversing down the both child nodes.
    - If too small, then the resulting nearest point may be suboptimal.

Computational Complexity and future improvement

- The computational complexity
  - Building the Tree: larger than O(NlogN) with O(N) space complexity
  - Searching the tree: ideally O(log N) but would be larger since the tree is not perfectly balanced and sometimes traversing the both nodes to find exact nearest neighbor point.
- Future improvement:
  - Reduce the computational complexity of building the tree, e.g. read the CSV table into ordered map and building the tree with the ordered data points (to decrease the complexity to find the median values), or use better algorithm to find the medial values (e.g. mergesort).