Database Project

Qiang Huang

April 26, 2016

Deadline: June 12, 23: 59 pm

1 Subject

The purpose of the project is to implement a well-known c-Approximate Nearest Neighbor (c-ANN) search algorithm named MEDRANK [1] using B^+-trees. The MEDRANK algorithm will be described later.

2 Programming Language and Experimental Environment

The programming language is limited to C/C++ and the project should be run under Linux. You can not use STL in this project, and you can not use any package from the Internet as well.

3 Problem Definition

Before we introduce the c-ANN search problem, we first define a distance metric named Euclidean distance. Given any two objects $o = (o_1, o_2, \ldots, o_d)$ and $q = (q_1, q_2, \ldots, q_d)$ in d-dimensional Euclidean space $\mathcal{R}^d$, the Euclidean distance between o and q, i.e., $\|o - q\|$, can be computed as follows:

$$\|o - q\| = \sqrt{\sum_{i=1}^{d} (o_i - q_i)^2}, \tag{1}$$

In this project, we use Euclidean distance as the distance metric for the c-ANN search. Given a database D of objects in $\mathcal{R}^d$, the c-ANN search problem is defined formally as follows.

Definition 1 (c-ANN search) *Given an approximation ratio $c > 1$, the c-ANN search problem is to construct a data structure which for any query $q \in \mathcal{R}^d$ finds an object $o \in D$ such that $\|o - q\| \leq c\|o^* - q\|$, where o^* is the nearest neighbor of q in D.*

4 The MEDRANK Algorithm

Since the regular paper [1] of the MEDRANK algorithm is a bit complicated, here we give an overview of the algorithm and its pseudo-code of MEDRANK for c-ANN search using B^+-trees.

The idea of MEDRANK came from voting. As we know, if an elector has more than a half votes from the voters, he/she will win this election. The MEDRANK algorithm is similar to an election. The algorithm first projects a database of objects and the queries on several random lines, and ranks the objects based on the distance of their projections to the projection of query. Each random line represents as a voter and each object is considered as an elector. An object will win a vote if for a specific random line, the MEDRANK

ALGORITHM 1: Indexing

```
Input: a database $D$ of $n$ data objects $\{o_1, o_2, \dots, o_n\}$ and the number of random projections $m$.
Output: None.
Generate a set of $m$ random projection vectors $\{\vec{a}_1, \vec{a}_2, \dots, \vec{a}_m\}$;
$L_j = \emptyset$ for $j \in \{1, 2, \dots, m\}$;
for $i = 1$ to $n$ do
    for $j = 1$ to $m$ do
        $c_{i,j} = i$;
        $v_{i,j} = \vec{a}_j \cdot \vec{o}_i$;
        $L_j = L_j \cup (c_{i,j}, v_{i,j})$;
    end
end
for $j = 1$ to $m$ do
    Sort $L_j$ in ascending order of the second component $v_{i,j}$;
    Index $L_j$ by a $B^+$-tree via bulk-loading and store it on the disk;
end
```

algorithm finds the object whose projection is closest to the projection of query. The MEDRANK algorithm repeatedly finds the closest object on each random line, and it stops until there is one object who wins more than a half votes. Then, the object wins this election and is regarded as the nearest neighbor of the query.

We now present a formal description of the MEDRANK algorithm. Given a database D of n objects, the MEDRANK algorithm first projects the objects on m random lines $\{L_1, L_2, \dots, L_m\}$ and indexes the objects on each lines with a B^+-tree. The pseudo-code of the indexing of MEDRANK is presented in Algorithm 1.

- Firstly, we randomly generate m random projection vectors $\{\vec{a}_1, \vec{a}_2, \dots, \vec{a}_m\}$, where each $\vec{a}_j$ is a d-dimensional vector whose entries are drawn independently and uniformly from the standard Normal distribution $N(0, 1)$ and we normalize each $\vec{a}_i$ to have unit length.
- Secondly, for each object o_i, we project it on m random line by doing an inner product, i.e., the projection of o_i on line L_j is $v_{i,j} = \vec{a}_j \cdot \vec{o}_i$. Then, each line L_j is a set of pairs $(c_{i,j}, v_{i,j})$, where $c_{i,j}$ is the object id of o_i on random line L_j.
- Finally, we sort each random line L_j in ascending order of the second component $v_{i,j}$, and index each L_j by a B^+-tree via bulk-loading and store it on the disk.

For a c-ANN query q, the MEDRANK algorithm is described as follows.

- For each line L_i, the algorithm first projects q and represents the projection as q_i. Then, the algorithm initializes two pointers h_i and l_i into L_i to specify the location of q_i via the search procedure of B^+-tree so that $v_{i,h_i} \leq q_i \leq v_{i,l_i}$.
- After specifying the location of the query projection on each random line L_i, the algorithm can simulate the voting procedure. Let S be a set of "seen objects" $c \in D$ when performing the search and Let f_c be the frequency of object c. We initialize $S = \emptyset$.
- The algorithm repeatedly finds the closest object c to q_i on each line L_i and move the pointer h_i or l_i up or down. For each object c we have seen, we increase its frequency f_c by 1. If there is one object whose frequency $f_c >$ MINFREQ $\cdot m$, the MEDRANK algorithm stops and returns this object as the nearest object for the query.

Notice that MINFREQ is a percentile parameter in a range $(0, 1)$, where the median rank corresponds to the setting of MINFREQ $= 0.5$. The result is more accurate if you set MINFREQ to a larger value than 0.5, at the cost of larger number of page I/Os and longer running time. The pseudo-code of the MEDRANK algorithm is presented in Algorithm 2.

ALGORITHM 2: MEDRANK

```
Input: a query object $q$ and a percentile MINFREQ
Output: $c \in S$ with the largest $f_c$.
for $i = 1$ to $m$ do
    $q_i = a_i * q$;
    initialize two pointers $h_i$ and $l_i$ into $L_i$ by the search procedure of $B^+$-tree so that $v_{i,h_i} \leq q_i \leq v_{i,l_i}$;
end
Let $S$ be a set of "seen objects" $c \in D$ and their frequencies $f_c$, initialize $S$ to $\emptyset$;
while $S$ has no object c such that $f_c >$ MINFREQ $\cdot m$ do
    for $i = 1$ to $m$ do
        if $|v_{i,h_i} - q_i| < |v_{i,l_i} - q_i|$ then
            $c = c_{i,h_i}$; $h_i = h_i - 1$;
        else
            $c = c_{i,l_i}$; $l_i = l_i + 1$;
        end
        if $c \notin S$ then
            add $c$ to $S$ and set $f_c = 1$;
        else
            $f_c = f_c + 1$;
        end
    end
end
return $c \in S$ with the largest $f_c$;
```

5 Dataset and Query Set

Please visit the website `http://yann.lecun.com/exdb/mnist/`. In this project, we use TRAINING SET IMAGE FILE (train-images-idx3-ubyte.gz) as dataset, and select the first 100 queries from TEST SET IMAGE FILE (t10k-images-idx3-ubyte.gz) as query set. Please read the format of the files on the website and extract the dataset and query set from the corresponding .gz files.

Please rename the dataset as `Mnist.ds`, where the cardinality is `n = 60,000` and the dimensionality of the data objects is `d = 784`. Accordingly, please rename the query set as `Mnist.q`, where the number of queries are `qn = 100`. The input format of the dataset and query set (both are text files) is described as follows:

```
1 object$_{1,1}$ object$_{1,2}$ ...  object$_{1,d}$
2 object$_{2,1}$ object$_{2,2}$ ...  object$_{2,d}$
...
N object$_{N,1}$ object$_{N,2}$ ...  object$_{N,d}$
```

Notice that each `object`$_{i,j}$ is the value of object `i` in `j`-th dimension, where `i` $\in \{1, 2, ..., N\}$ and `j` $\in \{1, 2, ..., d\}$. For the dataset `Mnist.ds`, `N` is the number of data objects `n`, while for the query set `Mnist.q`, `N` stands for the number of queries `qn`.

6 Implementation Requirements

There are several requirements for the implementation:

- Please notice that you must implement the B^+-tree and the MEDRANK algorithm by yourselves. If I find the implementation of B^+-tree or the MEDRANK algorithm is copied from other groups or copied from the Internet, the score of the project is 0.

- We limit the number of random projections $m = 50$, the percentile parameter MINFREQ = 0.5 and the page size B = 1 KB.

- In order to make a fair comparison for your implementations, you should report the index size and indexing time for building the B^+-tree, and evaluate **the average value** of overall ratio, I/O cost and running time of the MEDRANK algorithm when answering a set of c-ANN queries. The four evaluations are described as follows.

  - **Index Size and Indexing Time.** The index size is defined as the size of index generated by the MEDRANK algorithm. We also consider the indexing time to assess the time efficiency of building the index.
  - **Overall Ratio.** The overall ratio is used to measure the accuracy of a method. For the c-ANN search, it is defined as $\frac{\|o-q\|}{\|o^*-q\|}$, where o is the object returned by MEDRANK, and o^* is the exact nearest object. Intuitively, a smaller overall ratio means a higher accuracy.
  - **I/O Cost.** The I/O cost is defined as the number of pages to be accessed in order to answer a c-ANN query.
  - **Running Time.** The running time is defined as the wall-clock time for the MEDRANK algorithm to answer a c-ANN query.

7 Submission

All the files should be compressed into a zip file, which is named as "`groupID_Name_DBproject`", where `groupID` is your group id and `Name` is the name of your group leader. Please submit the zip file to the website `ftp://121.40.86.26/database`. The files you should submit are described as follows:

- **README.** Please write the steps to describe how to run the project. Please ensure the way is correct and the description is clear.

- **Source Codes.** Please create a directory "`/src`" to store the source codes of the project.

- **Dataset and Query Set.** Please create a directory "`/data`" which is used to store the dataset "`Mnist.ds`" and query set "`Mnist.q`". Please ensure the program can correctly read the dataset and query set from this directory. Since the dataset is large, please **DO NOT** submit the dataset and query set to the website.

- **Makefile.** The Makefile is used to compile the project. Please generate a executable program named `medrank` after compiling. The program should be run by the following command:

```
./medrank -n 60000 -d 784 -qn 100 -ds ./data/Mnist.ds -qs ./data/Mnist.q
```

- **Experimental Report.** You can write the report in Chinese or in English, but the format of the report should be a pdf file. The experimental report is named as "`groupID_report`", which consists of the following informations:

  - The Purpose of The Experiment
  - Experimental Environment
  - The Experimental Principle
  - Implementation Details (Please avoid copying your source codes.)
  - Experimental Results (Please add some analysis about the results.)
  - Conclusion
  - The Group Information and The Division of Each Member (Please write a list to describe what each group member did in the project.)

References

[1] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 301–312. ACM, 2003.