EFANNA:an Extreme Fast Approximate Nearest Neighbor search Algorithm framework based on kNN graph - User manual

Main author:Cong Fu, 15267003518@163.com
Deng Cai, dengcai@cad.zju.edu.cn
Contributor: Cong Fu, Deng Cai, Jinghao Lin[*],
Xiuye Gu[†]and Xiaoshuang, Zhang[‡]

November 29, 2016

1 Introduction

EFANNA is used for fast **Approximate Nearest Neighbor Search** (ANNS) problem based on a **Approximate** k **Neatest Neighbor Graph** (approximate kNN graph).

ANNS problem can be defined as follows: Given a metric space M, a base point set $D = \{p_1, p_2, p_3, ..., p_n\}$, a number of nearest neighbors of given new query point q in M should be returned quickly. However, searching for exact nearest neighbors with large scale and high dimensional data is too slow, and many proposed algorithms are even slower than brute-force search. Therefore, ANNS has drawn more and more attention. Given a number k, instead of finding the exact k nearest neighbor of q, ANNS algorithms search for the most possible k nearest neighbor for q. In other words, some points among the returned k are not the true top k neighbors. Due to sacrificing some accuracy, ANNS algorithms can achieve very high speed. EFANNA is much faster than all the previously proposed algorithms on ANNS.

EFANNA carries out ANNS based on a approximate kNN graph. A k nearest neighbor graph is a big table, For each point p in D, there is an entry recording k nearest neighbor in D in metric space M. However, it's also time costing to build an exact kNN graph. EFANNA can build approximate kNN graph quickly and outperforms previous algorithms, too.

So far EFANNA is written in C++ and MATLAB interfaces have been provided. Interfaces in other languages like Python and MATLAB will be provided

[*]fenixlin@yandex.com
[†]gxy0922@zju.edu.cn
[‡]zxs19930207@gmail.com

soon. EFANNA now supports SSE instructions acceleration. OpenMP and SIMD have be supported for speed-up.

2 Getting Started

2.1 Prerequisites

Efanna was developed and tested on 64-bit Linux and C++11. So far, we support GNU C++($\geq$ 4.9). And we use the cpu timer in boost library to test the performance. You may need to install the boost timer on Linux:

```
sudo apt-get install libboost-timer-dev libboost-system-dev
```

2.2 Building the project

The implementation of efanna algorithms is in the directory **algorithm**, and some samples showing how to use the API are in the directory **samples**. To build the project, you need to install some you may need go to the root directory of efanna and type:

```
cd efanna/
make
```

2.3 Running efanna samples

If you have complie the efanna project successfully. The executable file of the sample code will be in the directory **samples**. These samples shows how to use the API of efanna.

2.3.1 Index building

Efanna index including mainly two parts: tree index and graph index. We provide API **buildIndex()** to build them at one time.

First of all, you should create an **FIndex** object with the desired parameter setting of your desire.

The **Findex** object needs three parameters: data set matrix, distance type class and specific index parameter. The specific index parameters is a struct, whose name indicates the initial index type(only KD-tree currently). The parameters of it includes a binary flag indicating if use Reverse NN optimization(default is true), the total number of trees, the conquer-to-level (CL), number of iteration (IT), number of check (C), pool size (L), number of neighbors of each point (K, the 'width' of the returned K-NN graph), the number of trees for graph building(TB) and minimal pool size (S). And C, L, K, S and the R-NN flag is graph related. And there are some tips:

- $K \leq C \leq L$
- $TB \leq T$
- When TB, IT, CL, C, L and S is larger, The resulting K-NN graph is more accurate but slower.

You can see the sample code **efanna_index_buildall.cc** for more details.

To make the interfaces more flexible, you can build the tree and graph separately. We provide member functions of **FIndex** – **buildIndex()** to build the graph (since the graph building depends on tree initialization in our algorithms, there will always be tree construction before graph building.) and **buildTrees()** to build trees only. The number of trees is determined by the parameter T.

Of course we provide interfaces like **loadTrees(), saveTrees(), loadGraph()** and **saveGraph()**. They are naive and latter we will serialize those I/O interfaces.

2.3.2 K-NN search

Before you carry out K-NN search with prebuilt efanna index, you should set search-related parameters first. We provide **setSearchParams** to do this. It's also **FIndex**'s member function. And it needs 5 parameters – number of search epoches (SI), search pool size SP, search extension factor SE, the number of search trees(ST) and search method.

- $ST \leq T$, *i.e.* the number of search trees shouldn't exceed the number of trees you built.
- Currently, there are only two search methods, and we number them 0 1.
- Search method 0 works better on a 'narrow' graph (*e.g.* 10-NN graph). And SP is usually six times larger or more than SE. An empirical setting of SI is 4 for method 0.
- Search method 1 works better with a 'wider' graph (*e.g.* 100-NN graph). SE is usually equal to SP. And when SI, SP, SE are larger, the search recall is higher.
- Both methods work better with more trees.

2.3.3 Data I/O

We do not provide unified data I/O interfaces because the data file format varies from data set to data set. To use efanna, you should create a **Matrix** object which is designed for our data management.

The first thing you should do is to analyze you data file and load them into a big 1-d array. The points (*i.e.* features or vectors of data points) is continuously placed in the array. And then you can initialize the Matrix object with the total number of point, the dimension of the data, the pointer of the 1-d array.

You should be very carefully with data alignment when you want to use SIMD for speed-up (*e.g.* SSE or AVX instructions). For SSE, the vector length of one point should be a multiple of 4. For AVX, the vector length should be a multiple of 8. See the **load_data(...)** function in any sample file for details.

3 matlab interface

Code contributor Jinghao Lin [1]

3.1 Prerequisites

Alongside with the prerequisites of efanna from section 2.1, matlab 2010b for linux or later versions is required to run the matlab interface. Older versions and other platforms are temporarily not tested and the availability is not guaranteed.

3.2 Building the interface

Currently matlab interfaces needs to build separately from Efanna project. To use matlab interface, run the following command first.

```
cd efanna/matlab
mex CXXFLAGS="\$CXXFLAGS -std=c++11 -O3 -march=native -fopenmp -Wall
    -lboost_timer -lboost_system" LDFLAGS="\$LDFLAGS -fopenmp"
    findex.cc -I../ -largeArrayDims
```

If successfully built, you can see a file named findex.mexa64 under path efanna/matlab

3.2.1 Frequently asked questions

Due to differences of different versions and configurations of matlab installation, following problems may occur:

Error redeclaration of C++ built-in type 'char16_t' occurs in compilation

For this problem, try uncommenting lines starts with "define" and "undef" at the beginning of BOTH "findex.cc" and "handle_wrapper.hpp". Then compile again

Error related to mablab root/sys/os/glnxa64/libstdc++.so.6 occurs in runtime.

This error typically showed as version 'CXXABI_1.x.x' not found (required by ...), OR, version 'GLIBCXX_3.x.x' not found (required by ...)

A few methods may be useful:

- Try export LD_PRELOAD=$LD_PRELOAD:/usr/lib/x86_64-linux-gnu/libstdc++.so.6 , you may substitute with your own path of libstdc++.so.6 from your gcc compilers library path

[1] fenixlin@yandex.com

- Try export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/x86_64-linux-gnu/libstdc++.so.6 , you may substitute with your own path of libstdc++.so.6 from your gcc compilers library path
- directly substitute /usr/local/MATLAB/your version/sys/os/glnx64/libstdc++.so.6 with your compiler's libstdc++.so.6

For more details and further updates, please notice README.md under efanna/matlab folder on project github (also available from project source).

3.3 Running matlab examples

We offered a few examples under efanna/matlab/samples, which can do the same job as C++ examples under efanna/samples.

3.4 Index building & K-NN search

We provide a class efanna, including all methods of FIndex object in section 2.3.1 and 2.3.2, with same location and order of parameters. More details about parameters can be seen from sections above and README.md.

Additionaly, a sparse matrix specifying the graph connectivity will be return as the result of build_graph() and load_graph(). Also as can be seen from the third place in parameter list, a parameter for specifying distance method is provided. Currently only L2 distance is supported, and more will be provided later.

3.5 Data I/O

Data can be feed directly as matlab matrix, where every row represents a data point with every columns for a feature. Also the tool reading *.fvecs file into matlab matrix provided by Konda Reddy Mopuri[2] is attached to our project.

[2]https://cn.mathworks.com/matlabcentral/fileexchange/45340-display-sift-in-2d/content/Display_SIFT/fvecs_read.m