LSHBOX 0.1
User Manual

Zhifeng Xiao, xzf@whu.edu.cn

Gefu Tang, tanggefu@gmail.com

June 30, 2014

Chapter 1

Introduce

LSHBOX is a package that provides a randomized solution for the high-dimensional near neighbor problem in a variety of programming languages. After preprocessing the data set, LSHBOX answers queries, typically in sub-linear time, with each near neighbor being reported with a certain probability.

Chapter 2

Compilation

LSHBOX is written in the C++ programming language. And it also can be easily used in many contexts through the Python and MATLAB bindings provided with the library.

In order to make LSHBOX simple and easy to use, the library don't need to compile. You only need to add the include directory or modify the program search path, then you can use this library directly in C, C++, Python or MATLAB.

You can use CMAKE to build some tools for the test of this library.

In some cases, if you want or need to compile it by yourself with Python and MATLAB, please delete the comment of the last two lines in file "CMakeLists.txt", and you will find the compiling progress of python must rely on Boost library or some part of this library. For more detailed information, you can view the document "./python/README".

Chapter 3

Usage

This chapter contains small examples of how to use the LSHBOX library from different programming languages (C++, Python and MATLAB).

- **C++**

```
/**
 * @file itqlsh-run.cpp
 *
 * @brief Example of using Iterative Quantization for L2 distance.
 */
#include <lshbox.h>
```

```cpp
int main(int argc, char const *argv[])
{
    typedef float DATATYPE;
    std::cout << "LOADING DATA ..." << std::endl;
    lshbox::timer timer;
    lshbox::Matrix<DATATYPE> data("audio.data");
    std::cout << "LOAD TIME: " << timer.elapsed() << "s." << std::endl;
    std::cout << "CONSTRUCTING INDEX ..." << std::endl;
    timer.restart();
    std::string file = "itq.lsh";
    bool use_index = false;
    lshbox::itqLsh<DATATYPE> mylsh;
    if (use_index)
    {
        mylsh.load(file);
    }
    else
    {
        lshbox::itqLsh<DATATYPE>::Parameter param;
        param.M = 521;
        param.L = 5;
        param.D = data.getDim();
        param.N = 8;
        param.S = 100;
        param.I = 50;
        mylsh.reset(param);
        mylsh.train(data);
    }
    mylsh.save(file);
    std::cout << "CONSTRUCTING TIME: " << timer.elapsed() << "s.";
    std::cout << std::endl << "LOADING BENCHMARK ..." << std::endl;
    timer.restart();
    lshbox::Matrix<DATATYPE>::Accessor accessor(data);
    lshbox::Metric<DATATYPE> metric(data.getDim(), L1_DIST);
    unsigned K = 10;
    unsigned Q = 10;
    lshbox::Scanner<lshbox::Matrix<DATATYPE>::Accessor> scanner(
        accessor,
        metric,
        K,
        std::numeric_limits<float>::max()
    );
    std::cout << "LOADING TIME: " << timer.elapsed() << "s." << std::endl;
    std::cout << "RUNING QUERY ..." << std::endl;
    for (int I = 0; i != Q; ++i)
    {
        std::cout << "----- QUERY " << i+1  << " -----" << std::endl;
        scanner.reset(data[i]);
        mylsh.query(data[i], scanner);
        std::vector<std::pair<unsigned, float> > result;
        result = scanner.topk().getTopk();
        for (auto it = result.begin(); it != result.end(); ++it)
        {
            std::cout << it->first << ", " << it->second << std::endl;
        }
        std::cout << "Frequency : " << scanner.cnt() << std::endl;
    }
}
```

You can get the sample dataset 'audio.data' from http://www.cs.princeton.edu/cass/audio.tar.gz, if the link is invalid, you can also get it from here.

You can run the following program for a complete test in C++

- tools/create_test_data.cpp
- tools/scan_run.cpp
- tools/rbslsh_test.cpp
- tools/rhplsh_test.cpp
- tools/thlsh_test.cpp
- tools/psdlsh_test.cpp
- tools/shlsh_test.cpp
- tools/itqlsh_test.cpp

- Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pylshbox_example.py
import pylshbox
import numpy
# prepare test data
float_mat = numpy.random.rand(100000, 192)
float_query = float_mat[1,:]
unsigned_mat = numpy.int32(float_mat*5)
unsigned_query = unsigned_mat[1,:]
# Test rbsLsh
rbs_mat = pylshbox.rbslsh()
rbs_mat.init_mat(unsigned_mat.tolist(), '', 521, 5, 20, 5)
```

```
result = rbs_mat.query(unsigned_query.tolist(), 1)
indices, dists = result[0],result[1]
# Test rhpLsh
rhp_mat = pylshbox.rhplsh()
rhp_mat.init_mat(float_mat.tolist(), '', 521, 5, 6)
result = rhp_mat.query(float_query.tolist(), 2, 10)
indices, dists = result[0],result[1]
# Test thLsh
th_mat = pylshbox.thlsh()
th_mat.init_mat(float_mat.tolist(), '', 521, 5, 12)
result = th_mat.query(float_query.tolist(), 2, 10)
indices, dists = result[0],result[1]
# Test psdlsh with param.T = 1
psdL1_mat = pylshbox.psdlsh()
psdL1_mat.init_mat(float_mat.tolist(), '', 521, 5, 1, 5)
result = psdL1_mat.query(float_query.tolist(), 2, 10)
indices, dists = result[0],result[1]
# Test psdlsh with param.T = 2
psdL2_mat = pylshbox.psdlsh()
psdL2_mat.init_mat(float_mat.tolist(), '', 521, 5, 2, 0.5)
result = psdL2_mat.query(float_query.tolist(), 2, 10)
indices, dists = result[0],result[1]
# Test shLsh
sh_mat = pylshbox.shlsh()
sh_mat.init_mat(float_mat.tolist(), '', 521, 5, 4, 100)
result = sh_mat.query(float_query.tolist(), 2, 10)
indices, dists = result[0],result[1]
# Test itqLsh
itq_mat = pylshbox.itqlsh()
itq_mat.init_mat(float_mat.tolist(), '', 521, 5, 8, 100, 50)
result = itq_mat.query(float_query.tolist(), 2, 10)
indices, dists = result[0],result[1]
```

- MATLAB

```
% lshbox_example.m
% prepare test data
dataset = rand(128,100000);
testset = dataset(:,1:10);
% Test rhplsh
param_rhp.M = 521;
param_rhp.L = 5;
param_rhp.N = 6;
[indices, dists] = rhplsh(dataset, testset, param_rhp, '', 2, 10)
```

```
% Test thlsh
param_th.M = 521;
param_th.L = 5;
param_th.N = 12;
[indices, dists] = thlsh(dataset, testset, param_th, '', 2, 10)
% Test psdlsh with param_psdL1.T = 1
param_psdL1.M = 521;
param_psdL1.L = 5;
param_psdL1.T = 1;
param_psdL1.W = 5;
[indices, dists] = psdlsh(dataset, testset, param_psdL1, '', 1, 10)
% Test psdlsh with param_psdL2.T = 2
param_psdL2.M = 521;
param_psdL2.L = 5;
param_psdL2.T = 2;
param_psdL2.W = 0.5;
[indices, dists] = psdlsh(dataset, testset, param_psdL2, '', 2, 10)
% Test shlsh
param_sh.M = 521;
param_sh.L = 5;
param_sh.N = 4;
param_sh.S = 100;
[indices, dists] = shlsh(dataset, testset, param_sh, '', 2, 10)
% Test itqlsh
param_itq.M = 521;
param_itq.L = 5;
param_itq.N = 8;
param_itq.S = 100;
param_itq.I = 50;
[indices, dists] = itqlsh(dataset, testset, param_itq, '', 2, 10)
```

Have you ever find the empty string used in the Python and MATLAB code? In fact, they can be used to save the index through pass a file name. Like the following, you will find the next query speed faster than the first, because there is no re-indexing.

- Python

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# pylshbox_example2.py
import pylshbox
import numpy
import time
# prepare test data
```

```
float_file = 'audio.data'
float_query = numpy.random.rand(192)
# Test itqLsh
# First time, need to constructing index. About 1.5s.
start = time.time()
itq_file = pylshbox.itqlsh()
itq_file.init_file(float_file, 'pyitq.lsh', 521, 5, 8, 100, 50)
result = itq_file.query(float_query.tolist(), 2, 10)
print 'Elapsed time is %f seconds.' % (time.time() - start)
# Second time, no need to re-indexing. About 0.05s.
start = time.time()
itq_file2 = pylshbox.itqlsh()
itq_file2.init_file(float_file, 'pyitq.lsh', 521, 5, 8, 100, 50)
result = itq_file2.query(float_query.tolist(), 2, 10)
print 'Elapsed time is %f seconds.' % (time.time() - start)
```

- MATLAB

```
% lshbox_example2.m
% prepare test data
dataset = rand(128,500000);
testset = dataset(:,1:10);
% Test itqlsh
param_itq.M = 521;
param_itq.L = 5;
param_itq.N = 8;
param_itq.S = 100;
param_itq.I = 50;
% First time, need to constructing index. About 10s.
tic;
[indices, dists] = itqlsh(dataset, testset, param_itq, 'itq.lsh', 2, 10);
toc;
% Second time, no need to re-indexing. About 2s.
tic;
[indices, dists] = itqlsh(dataset, testset, param_itq, 'itq.lsh', 2, 10);
toc;
```

Chapter 4

Algorithm

LSHBOX is based on many approximate nearest neighbor schemes, and the following is a brief description of each algorithm and its parameters.

- **Locality-Sensitive Hashing Scheme Based on Random Bits Sampling**

✧ Reference

P. Indyk and R. Motwani. Approximate Nearest Neighbor - Towards Removing the Curse of Dimensionality. In Proceedings of the 30th Symposium on Theory of Computing, 1998, pp. 604-613.

A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. Proceedings of the 25th International Conference on Very Large Data Bases (VLDB), 1999.

✧ Parameters

```
struct Parameter
{
    /// Hash table size
    unsigned M;
    /// Number of hash tables
    unsigned L;
    /// Dimension of the vector
    unsigned D;
    /// Binary code bytes
    unsigned N;
    /// The Difference between upper and lower bound of each dimension
    unsigned C;
};
```

✧ Implementation

```
#include <lshbox/rbslsh.h>
```

✧ Notice

According to the second assumption in the paper, all coordinates of points in P are positive integer. Although we can convert all coordinates to integers by multiplying them by a suitably large number and rounding to the nearest integer, but I think it is very fussy, What's more, it often gets criticized for using too much memory when in a larger range of data. Therefore, it is recommended to use other algorithm.

- **Locality-Sensitive Hashing Scheme Based on Random Hyperplane**

✧ Reference

Charikar, M. S. 2002. Similarity estimation techniques from rounding algorithms. In Proceedings of the Thiry-

Fourth Annual ACM Symposium on theory of Computing (Montreal, Quebec, Canada, May 19 - 21, 2002). STOC '02. ACM, New York, NY, 380-388. DOI= http://doi.acm.org/10.1145/509907.509965

✧ Parameters

```
struct Parameter
{
    /// Hash table size
    unsigned M;
    /// Number of hash tables
    unsigned L;
    /// Dimension of the vector
    unsigned D;
    /// Binary code bytes
    unsigned N;
};
```

✧ Implementation

```
#include <lshbox/rhplsh.h>
```

● Locality-Sensitive Hashing Scheme Based on Thresholding

✧ Reference

Zhe Wang, Wei Dong, William Josephson, Qin Lv, Moses Charikar, Kai Li. Sizing Sketches: A Rank-Based Analysis for Similarity Search. In Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems . San Diego, CA, USA. June 2007.

Qin Lv, Moses Charikar, Kai Li. Image Similarity Search with Compact Data Structures. In Proceedings of ACM 13th Conference on Information and Knowledge Management (CIKM), Washington D.C., USA. November 2004.

✧ Parameters

```
struct Parameter
{
    /// Hash table size
    unsigned M;
    /// Number of hash tables
    unsigned L;
    /// Dimension of the vector
    unsigned D;
    /// Binary code bytes
    unsigned N;
    /// Upper bound of each dimension
    float Max;
    /// Lower bound of each dimension
    float Min;
};
```

✧ Implementation

```
#include <lshbox/thlsh.h>
```

● **Locality-Sensitive Hashing Scheme Based on p-Stable Distributions**

✧ Reference

Mayur Datar , Nicole Immorlica , Piotr Indyk , Vahab S. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, Proceedings of the twentieth annual symposium on Computational geometry, June 08-11, 2004, Brooklyn, New York, USA.

✧ Parameters

```
struct Parameter
{
    /// Hash table size
    unsigned M;
    /// Number of hash tables
    unsigned L;
    /// Dimension of the vector
    unsigned D;
    /// Index mode, you can choose 1(CAUCHY) or 2(GAUSSIAN)
    unsigned T;
    /// Window size
    float W;
};
```

✧ Implementation

```
#include <lshbox/psdlsh.h>
```

● **Spectral Hashing**

✧ Reference

Y. Weiss, A. Torralba, R. Fergus. Spectral Hashing. Advances in Neural Information Processing Systems, 2008.

✧ Parameters

```
struct Parameter
{
    /// Hash table size
    unsigned M;
    /// Number of hash tables
    unsigned L;
    /// Dimension of the vector
    unsigned D;
    /// Binary code bytes
    unsigned N;
    /// Size of vectors in train
    unsigned S;
};
```

✧ Implementation

```
#include <lshbox/shlsh.h>
```

- **Iterative Quantization**

✧ Reference

Gong Y, Lazebnik S, Gordo A, et al. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2013, 35(12): 2916-2929.

✧ Parameters

```
struct Parameter
{
    /// Hash table size
    unsigned M;
    /// Number of hash tables
    unsigned L;
    /// Dimension of the vector
    unsigned D;
    /// Binary code bytes
    unsigned N;
    /// Size of vectors in train
    unsigned S;
    /// Training iterations
    unsigned I;
};
```

✧ Implementation

```
#include <lshbox/itqlsh.h>
```

✧ Notice

According to the test, Iterative Quantization performance very good and is superior to other schemes, it can get better query accuracy with minimum cost.

Please don't hesitate to contact me if you have any questions.