Yael Documentation

Release 300

Herve Jegou & Matthijs Douze

September 13, 2012

CONTENTS

CHAPTER

ONE

WHAT IS THIS?

This is a tutorial for first steps with Yael. Most of the remarks and details here will be just but useless for skilled programmers, as the structure of the library should be clear enough. The documentation related to Python should however interest any programmer wanting to use this interface.

Yael is a library for performing efficient basic operations, in particular kmeans and exhaustive nearest neighbor search function. It offers three interfaces:

- C,
- Python,
- Matlab (Octave extension is included, but not fully checked at each release).

The library has been tested under different architectures, in particular

- Linux 32 bits: Fedora Core 11
- Linux 64 bits: Fedora Core 10, Fedora Core 11, Ubuntu Karmic, Debian 4.1.2-25
- Mac OS X.

The library has not been packaged nor tested for Microsoft Windows. Some people have however adapted some previous version of the library on this operating system.

The C API is object-oriented in spirit whenever applicable, with constructors and destructors for each structure. All functions are re-entrant, but unless otherwise specified, they are not multi-threaded. Threading is assumed to occur at a higher level. In particular, the python interface offers a very simple way to parallelize function calls.

CHAPTER

TWO

GETTING STARTED

2.1 Getting SVN version and content

Retrieve the last archive file on the gforge INRIA web server. Windows users should probably use an older version, as we do not support Windows (existing versions are external contributions which are not maintained).

You may be interested in the last in obtaining the latest version of Yael. For registered users, it can be retrieved from the SVN as follows:

```
svn checkout svn+ssh://mygforgelogin@scm.gforge.inria.fr/svn/yael/trunk yael
```

This performs the installation of the regular Yael branch of the project into the directory Yael. This installation directory is referred to as YAELROOT in the following. There is also an anonymous access, which is provided by using one of the following commands:

```
svn checkout --username anonsvn https://scm.gforge.inria.fr/svn/yael
```

The password for anonymous login is ‘anonsvn’.

The library is organized as follows:

```
YAELROOT/                root directory
YAELROOT/LICENCE         licensing information
YAELROOT/Makefile        generic Makefile
YAELROOT/makefile.inc    configuration file generated by ‘onfigure.sh
YAELROOT/configure.sh    configuration program with basic auto-detection procedures.
YAELROOT/README          subset of this getting started manual
YAELROOT/doc/            documentation: getting started and reference manuals
YAELROOT/matlab/         matlab interface
YAELROOT/progs/          executable programs: front-end and utilities
YAELROOT/test/           sample test programs for various functions
YAELROOT/yael/           core yael directory
```

2.2 Pre-requisites

The library requires the following software/libraries to be installed. Some of them are related to the non-core interfaces (python and matlab) and are not strictly mandatory.

Blas and Lapack. Any implementation should work as it is wrapped with the Fortran calling conventions. It might be required to adjust the location of these libraries in the `makefile.inc` file generated by the `configure.sh` script. The `LD_LIBRARY_PATH` environment variable may also need to be set accordingly.

Python-dev. This package contains the include files of the Python-C API. Note that having python installed on your machine does not necessarily mean that the development kit is installed as well. If only python is installed, not python-dev, you will get an error saying that the file `Python.h` can not be found.

swig is required to create the python interface.

Matlab is required only for the matlab Yael interface. It is not compiled by default.

doxygen is used to generate the reference manual (HTML) from the source files.

sphinx and **pdflatex** are used to re-generate this getting started manual.

These two last pre-requisites are required by the Python interface and are not strictly mandatory. If you are not interested by this Python interface and that you don't have python-dev and swig installed, you should remove the target `_yael.so` target in the Makefile of the yael core directory.

2.3 Installation procedure

The source is compiled with hand-written Makefiles. All system-dependent options are defined in the toplevel `makefile.inc`. The makefiles can be called in their directory.

First execute:

```
./configure.sh
```

in the yael root directory. This generates the file `makefile.inc` using basic auto-detection procedures. For most configurations, these should be sufficient. However you might need to adjust the variables defined in `makefile.inc` to fit your local configuration.

The standard make tool is used to compile the library:

```
make
```

If everything goes fine, your should be able to compile the programs in progs and test.

At this point, only the core C and python library are compiled. If you need the Matlab interface, you have to compile the mex files [1]

```
cd matlab
make
```

In order to generate the reference manual and this tutorial, you should go into the doc subdirectory and execute the following command line:

```
cd doc
make
```

Remark: In order to use the 32 bit mode on MacOS (not recommanded), use the special flag `--mac32`:

```
./configure.sh --mac32
```

[1] The mex executable should be in the PATH. This is a typical error. Another one occurs when the `mex` utility of Octave has a higher priority than the Matlab command with the same name.

CHAPTER

THREE

C INTERFACE AND BASIC PROGRAMS

The C API is object-oriented whenever applicable, with constructors and destructors for each structure.

The include directory should be set to the yroot directory, so that a yael file is included using the prefix `yael/`. For instance, including the primitive for vectors is performed by:

```
#include <yael/vector.h>
```

The documentation of the functions and data structures is in the header files. It can be extracted and formatted by `doxygen`, run by invoking `make` in the `doc` subdirectory.

The best thing to do to have an operational Makefile and program is probably to look at the test files included in the `YAELROOT/test/` subdirectory.

3.1 Conventions

Vectors are represented as C arrays of basic elements. Functions operating on them are prefixed with:

- `ivec`: basic type is `int`
- `fvec`: basic type is `float`
- `dvec`: basic type is `double`
- `bvec`: basic type is `unsigned char`

Most of the functions are associated with `ivec` and `fvec` types. Vector sizes are passed explicitly, as long int's to allow for large arrays on 64 bit machines. Vectors can be free'd with `free()`:

```
/* Generate a random array using a thread-safe function */
long n = 100; \
seed = 666;
float * v = fvec_new_randn_r (n, seed);

double sum_v = fvec_sum (v, n);

/* free the vector */
free (v);
```

Arrays of vectors are stored contiguously in memory. As shown above, an array of n float vectors of dimension d is simply declared a pointer on float, as `float *fv`.

The *i*-th element of vector *j* of vector array *vf*, where $0 \leq i < d$ and $0 \leq j < n$ is:

```
vf[j * d + i]
```

It can also be seen as a column-major matrix of size $d * n$:.

Since the library is intended to be fast, 32-bit floating point numbers are preferred over 64-bit ones almost everywhere.

We acknowledge the influence of Fortran conventions in this design.

3.2 Multi-threading in Yael

In order to exploit multi-CPU and multi-core architectures, many functions in Yael execute in multiple threads. They take an additional parameter (`int nt`) that defines the number of threads to use.

The number of CPUs available on the machine is given by the `count_cpu()` function.

3.2.1 Multi-threading primitive

Most multi-threading operations in Yael are implemented via the function:

```
void compute_tasks (int n, int nt,
                    void (*task_fun) (void *arg, int tid, int i),
                    void *task_arg);
```

The function executes *n* tasks on *nthread* threads. For each task, the callback *task_fun* is called with `task_arg` as first argument, `i = 0..n-1` set to the task number, and `tid = 0..nt-1` set to the thread number.

Some operations are also multi-threaded via OpenMP pragmas. These are enabled only with the `--enable-openmp` option of the `./configure.sh` script, else they will run single-threaded.

3.2.2 Best practices

It is often advisable to perform multi-threading at the highest level possible. For example, with two nested loops:

```
for (i = 0 ; i < 1000 ; i++)
  for (j = 0 ; j < 1000 ; j++)
    expensive_operation (i, j);
```

It is more efficient (and often easier) to define a task as one outer loop. This reduces the number of synchronization barriers.

3.2.3 Multithreading and random numbers

Yael relies on the random number generators `rand()` and `lrand48()`. These can be seeded, which changes a global state variable. However, if the random number generators are called from multiple threads, the sequences get mixed and are not reproducible any more.

To avoid this, several Yael functions that use random generators have an additional parameter that sets a local random seed. They are often suffixed with `_r` (**r**eentrant), for example: `fvec_new_rand_r`, `ivec_new_random_idx_r`, `kmeans`.

CHAPTER

FOUR

PYTHON INTERFACE

The whole C API is exposed in Python using SWIG, hence the tc{.swg} files in the subdirectories. This allows to call C functions from Python more or less transparently.

4.1 Loading and using Yael

Assuming that the PYTHONPATH environment variable is set to Yael's installation root, importing the Yael interface and creating a new vector is done as:

```
from yael import yael
a = yael.fvec_new_0(5)
```

In order to shorten the call, one could also import the function in the current namespace, as

> from yael.yael import * a = fvec_new_0(5)

However, we do not advise to do so, in order to avoid function name conflicts when using other python libraries jointly with Yael.

4.2 Guidelines for the wrapping process

- for most of the objects, memory is **not** managed by Python. They must be free'd explicitly. The main exception is for vectors, which can be explicitly acquired by Python so that they are garbage-collected like a Python object
- arrays for simple types are called `ivec`, `fvec`, etc. Usage:
  - `a = ivec(4)` constructs an array of 4 ints, accessible in Python with `a[2]`, as one would expect. There is no bound checking: the Python object does not know about the size of the array (like with C pointers).
  - `a.cast()` returns an `int*` usable as a C function argument (most of the time, the cast is automatic, and `a` can be used when a function expects an `int *`).
  - if a C function returns an `int*`, `b = ivec.frompointer(x)` makes the Python `a[i]` valid to access C's `x[i]`.
  - `b.plus(2)` returns `x + 2` (pointer arithmetic).
  - `b = ivec.acquirepointer(x)` will, in addition, call `free(x)` when the Python object `b` is deleted. This function therefore ensures that `x` will be cleaned up by the Python garbage collector. Often, when a C function returns a newly allocated pointer `x`, it is advisable to immediately do `x=ivec.acquirepointer(x)`.
  - `a.clear(3)` clears out the 3 first elements of the vector.

   - if `c` is another `int*`, `a.copyfrom(c, 1, 2)` will copy 2 elements from `c` to `a` at offset 1 (ie. `a[1] = c[0]` and `a[2] = c[1]`).
   - `a.tostring(3)` returns a Python string with the 3 first elements of `a` as raw binary data. They can be used eg. in the `array` module.
   - similarly, `a.fromstring(s)` fills the first elements of `a` with raw values read from the string.
- all wrapped functions release Python's Global Interpreter Lock (to allow multithreaded execution), so Python API functions should not be called in C code.
- output arguments in the C code (their names end in `_out`) are combined with the function results tuples.

4.3 NumPy interface

If Yael is configured with `--enable-numpy`, arrays can be exchanged with Numpy arrays. This is done through a series of functions with self-explanatory names:

```
fvec_to_numpy
ivec_to_numpy
numpy_to_fvec
numpy_to_ivec
```

Arrays corresponding to Yael's `fvec``are of Numpy's ``dtype='float32'`. Moving from yael to numpy produces line vectors, that can be reshaped to matrices if needed.

These functions copy their arguments. To share the same data buffer between Yael and Numpy, suffix the function with `_ref`.

See the `test_numpy.py` program for an example usage.

4.4 Numpy interface (high level)

A few functions of Yael are also made available with pure Numpy arguments and return types. They are in the `ynumpy` module. They include:

```
knn
kmeans
fvecs_read ivecs_read siftgeo_read
```

All matrix arguments should be in C indexing, because numpy's support for Fortran-style indexing is close to unusable. Therefore, in the function documentation, all references to "columns" should become "lines". See `test_ynumpy.py` for an example.

4.5 ctypes interface

Arrays can also be exchanged with ctypes. This is done by converting pointers to integers. See `test_ctypes.py` for an example.

CHAPTER

FIVE

MATLAB

This chapter is dedicated to the Matlab interface. However, most of the comments here have their Octave equivalent.

5.1 Content of the Matlab interface

The Matlab interface of Yael is limited to functions that are not available in Matlab, ie. most basic matrix manipulation functions are readily available in Matlab or can be implemented trivially.

The functions currently provided are:

- `yael_kmeans`. Although there is a kmeans function in Matlab, that one is not very efficient. Moreover, it is not available in the core Matlab program, since it requires a specific toolbox.
- `yael_knn` is used to find the k nearest neighbors with respect to the Euclidean distance. Although for $k = 1$ Matlab does a good job, finding $k > 1$ neighbors requires the `sort` function, which is very inefficient when k is small compared to the number of vectors.
- `yael_L2sqr` computes all the square distances between two sets of vectors. Therefore, it computes $n_1 \times n_2$ distances.
- `yael_kmin` and `yael_kmax` compute the k smallest (or largest) values of a set of scalar. It is more efficient than sorting the data.
- `yael_fvecs_normalize` normalizes a set of vectors.
- `yael_gmm` learns a Gaussian mixture model (diagonal form).
- `yael_fisher` computes the Fisher Kernel representation of a set of features.

There are also several I/O functions, see below

5.2 Using the Yael interface

The functions are implemented by Mex-file, which requires to compile the Matlab interface of Yael (disabled by default). To do so, just type `make` in the `matlab` directory. You might obtain a warning about a version with the gcc version used. We never observed any trouble related to that point. In order for the functions to be found in matlab, you should include this subdirectory in the MATLAB path, for instance by setting the environment variable:

```
export MATLABPATH=$MATLABPATH:YAELROOT/matlab
```

in your shell configuration files (e.g., `~/.bash_profile` for bash) where `YAELROOT` should be set to fit your local configuration. Inside Matlab, the `addpath` function can be used for the same purpose.

Conveniently, Matlab and Yael use the same convention for matrix storage (column-major).

All the functions of the Yael-Matlab interface use single precision vectors, i.e., the Matlab equivalent of float. This is in contrast to Matlab's default double precision floating numbers. Therefore, one has to cast the input data to single precision when calling Yael functions, e.g., as:

```
[dis, ids] = yael_nn(single(v), single(q), 10)
```

CHAPTER

SIX

FILE EXCHANGE FORMAT

Yael uses a simple file format to store byte, integer or floating point vectors. Each vector is stored by

- writing its length `d` as an integer (4 bytes, little-endian)
- writing its `d` components in binary format.

Until now, we have used only single-precision floating point numbers. Therefore the size of storing an integer or a floating-point vector is simply `sizeof(int) + 4 * d` :math:‘ = 4 (d+1)‘, as both the dimension and the components require 4 bytes each. For byte vectors, it is similarly equal to `4+d`.

By convention we will call a file containing several vectors of the same kind ‘‘fvecfile’‘, ‘‘ivecfile” and ‘bvecfile’.

Matrices are stored as a concatenation of vectors, again without a header. A float matrix is therefore a fvecfile where all vectors are the same length. Its suggested filename extention is `.fvecs`.

Some particular high-level functions are defined for these files.

Remarks:

- Since there use no header, several files of same type can be concatenated using regular the `cat` Unix command to produce a file containing the concatenation of the vector.
- in the `progs` subdirectory, the utilities `xvecfile` (x= `f`, `i` or `b`) receive a xvecfile on standard input and produce a text output. Usage:

  ```
  cat myfile.fvecs | fvecfile
  ```

  or, equivalently:

  ```
  fvecfile < myfile.fvecs
  ```

- The Yael functions used to read these the files in C, Python and Matlab are called `Xvecs_read` (x= `f`, `i` or `b`). The prototype of the function depends on whether you call it from C, Python or Matlab. See the I/O interface in vector.h.

CHAPTER

SEVEN

TROUBLESHOOTING

Before reporting any trouble in installing the library, please ensure that the following points are correctly configured Most of the problems should be related to an incorrect configuration in `makefile.inc`.

- **Problems when linking**. The environment variable associated with dynamic library should be set to tc{yroot/yael}.

Architecture	yael library name	environment variable	command to check missing libs
Linux 32/64	`libyael.so`	`LD_LIBRARY_PATH`	`ldd libyael.so`
MacOS X	`libyael.dylib`	`DYLD_LIBRARY_PATH`	`otool -L libyael.dylib`

  Note that the path to the dynamic library can also be hardcoded in the code that used i, with `-Wl,-rpath,YAELROOT`. This is done for Python's interface.

- **Python**. The `PYTHONPATH` environment variable should point to YAELROOT. The `*_LIBRARY_PATH` variable does not need to be set.
- Segfault in Mexfile: MacOS 32/64 bits. If you have a 32 bits Matlab version with MacOS, then you should check that you have used the flag `--mac32` when configuring Yael.