SmallK: A Library for Nonnegative Matrix Factorization, Topic Modeling, and Clustering of Large-Scale Data

Release 1.6.2

BARRY DRAKE [1] AND STEPHEN LEE-URBAN [2]

Information and Communications Laboratory
Georgia Tech Research Institute
250 14th St. NW.
Atlanta, GA 30318

HAESUN PARK [3]

School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA 30332

Apr 18, 2018

[1]Electronic address: bldrake@cc.gatech.edu; Corresponding author
[2]Electronic address: stephen.urban@cc.gatech.edu
[3]Electronic address: hpark@cc.gatech.edu

CONTENTS

CHAPTER

ONE

ABOUT

SmallK is a high performance software package for constrained low rank matrix approximation via the nonnegative matrix factorization (NMF). Algorithms for NMF compute the low rank factors of a matrix producing two nonnegative matrices whose product approximates the original matrix. The role of NMF in data analytics has been as significant as the singular value decomposition (SVD). However, due to nonnegativity constraints, NMF has far superior interpretability of its results for many practical problems such as image processing, chemometrics, bioinformatics, topic modeling for text analytics and many more. Our approach to solving the NMF nonconvex optimization problem has proven convergence properties and is one of the most efficient methods developed to date.

1.1 Distributed Versions

Recently open sourced: MPI-FAUN! Both MPI and OPENMP implementations for MU, HALS and ANLS/BPP based NMF algorithms are now available. The implementations can run off the shelf or can be easily integrated into other source code. These are very highly tuned NMF algorithms to work on super computers. We have tested this software in NERSC as well OLCF cluster. The openmp implementation is tested on many different linux variants with intel processors. The library works well for both sparse and dense matrices.

Please visit MPI-FAUN text[1] for more information and source code.

1.2 Ground truth data for graph clustering and community detection

Community discovery is an important task for revealing structures in large networks. The massive size of contemporary social networks poses a tremendous challenge to the scalability of traditional graph clustering algorithms and the evaluation of discovered communities.

Please visit dblp ground truth data[2] to obtain the data.

For U.S. Patent data go test hybrid clustering of content and connection structure using joint NMF go to patent data[3] to view the readme.

1.3 Acknowledgements

This work was funded in part by the DARPA XDATA program under contract FA8750-12-2-0309. Our DARPA program manager is Mr. Wade Shen[4] and our XDATA Principal Investigator is Prof. Haesun Park[5] of the Georgia

[1] https://github.com/ramkikannan/nmflibrary
[2] https://github.com/smallk/smallk_data/tree/master/dblp_ground_truth
[3] https://github.com/smallk/smallk_data/tree/master/patent
[4] http://www.darpa.mil/staff/mr-wade-shen
[5] http://www.cc.gatech.edu/~hpark/

Institute of Technology. We would like to thank Rundong Du for the dblp ground truth data set and Dr. Ramakrishnan Kannan of ORNL for the MPI-FAUN! distributed code. Also, special thanks to Dr. Richard Boyd, Dr. Da Kuang, and Ashley Scripka-Beavers for their contributions to previous versions of this documentation and significant technical contributions. A final special thanks to Ethan Trewhitt, who created the Docker install and advised on how to use Sphinx and RTD for documentation.

1.4 Contact Info

For comments, questions, bug reports, suggestions, etc., contact:

Barry Drake
Research Scientist
Information and Communications Laboratory (ICL)
Information and Cyber Sciences Directorate (ICSD)
Georgia Tech Research Institute (GTRI)
75 5TH St. NW STE 900
ATLANTA, GA 30308-1018
barry.drake@gtri.gatech.edu

Stephen Lee-Urban
Research Scientist
Information and Communications Laboratory (ICL)
Information and Cyber Sciences Directorate (ICSD)
Georgia Tech Research Institute (GTRI)
75 5TH St. NW STE 900
ATLANTA, GA 30308-1018
stephen.lee-urban@gtri.gatech.edu

CHAPTER

TWO

INTRODUCTION

2.1 Background

High-dimensional data sets are ubiquitous in data science, and they often present serious problems for researchers. Our work in dimensionality reduction focuses on, but is not limited to, low rank approximations via nonnegative matrix factorization (NMF) [see Publications [1,2][6]]. NMF is a non-convex optimization problem with important applications in data and interactive visual analytics of high-dimensional data.

The impetus for this documentation is to provide a step-by-step procedure for the application of the theory to real-world large-scale data analytics problems. We have instantiated our research efforts in a software framework that includes high-level driver code via Python and a simple command line interface, SmallK, which hides most of the details of the input parameters. Our low-level code, also usable from the command line, is written in C++, which provides efficient NMF algorithms. The algorithms discussed herein have numerous practical applications; this document and the tutorials[7] will provide the information required to quickly begin real work.

Below is a brief description of our fundamental research on NMF algorithms. Following the brief motivational introduction to the NMF are detailed installation instructions for the SmallK software library.

2.2 Constrained low rank approximations and NMF

Algorithms that enable dimension reduction and clustering are two critical areas in data analytics and interactive visual analysis of high-dimensional data. A low rank approximation framework has the ability to facilitate faster processing times and utilize fewer resources. These approximations provide a natural way to compute only what we need for significant dimension reduction, and are analogous to singular value decomposition (SVD) and principal component analysis (PCA). Our algorithm framework also works efficiently for clustering since clustering can be viewed as a specific way of achieving a low rank approximation so that the cluster structure of the data is well represented in a few basis vectors.

Matrix low rank approximations such as the SVD have played a key role as a fundamental tool in machine learning, data mining, and other areas of computational science and engineering. The NMF has recently emerged as an important constrained low rank approximation method as well. A distinguishing feature of the NMF is the requirement of nonnegativity: NMF is considered for high-dimensional and large scale data in which the representation of each element is inherently nonnegative, and it seeks low rank factor matrices that are constrained to have only nonnegative elements. There are many examples of data with a nonnegative representation. In a standard term-frequency encoding, a text document is represented as a vector of nonnegative numbers since each element represents the number of appearances of each term in each document. In image processing, digital images are represented by pixel intensities, which are nonnegative. In the life sciences, chemical concentrations or gene expression levels are naturally represented as nonnegative data.

[6] http://smallk.github.io/publications/
[7] http://smallk.github.io/documentation/tutorials/

Our algorithm framework utilizes various constraints on the non-convex optimization problem that gives rise to the nonnegative factors. With these various constraints NMF is a versatile tool for a large variety of data analytics problems. NMF algorithms have been an active area of research for several years. Since much of the data for many important problems in numerous domains is nonnegative NMF is the correct computational model for mining and/or integrating information from such data. NMF also offers enhanced interpretation of results since nonnegativity of the data is preserved.

2.3 SmallK Overview

The SmallK library provides routines for low rank matrix approximation via nonnegative matrix factorization (NMF). The term "nonnegative matrices" means that for a given matrix all of it's elements are greater than or equal to zero, which we express as ≥ 0.

Given a nonnegative matrix $\boldsymbol{A}$, the SmallK software computes nonnegative matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ such that $\boldsymbol{A} \cong \boldsymbol{W}\boldsymbol{H}$

The matrix $\boldsymbol{A}$ has m rows and n columns and can be either sparse or dense. $\boldsymbol{W}$ has m rows and k columns, and $\boldsymbol{H}$ has k rows and n columns. The value of k is an input parameter to the approximation routines; typically k $\ll$ m and k $\ll$ n. Where k is the reduced rank of the low rank approximation and, in applications, it represents, for example, the reduced dimension in dimension reduction, number of clusters for clustering various data sets, or the number of topics in topic discovery.

NMF algorithms seek to approximate a matrix $\boldsymbol{A}$ by the product of two much smaller matrices $\boldsymbol{W}$ and $\boldsymbol{H}$. The idea is to choose the smallest value of k (width of $\boldsymbol{W}$ and height of $\boldsymbol{H}$) that gives an acceptable approximation error. Due to the nonconvex nature of the optimization problem associated with finding $\boldsymbol{W}$ and $\boldsymbol{H}$, they can only be approximated after an NMF algorithm satisfies a convergence criterion to a local minimum. Thus, the minimization of the objective function proceeds iteratively, attempting to reach a stationary point, which is the best possible solution. As the iterations proceed, the SmallK code computes a metric that estimates the progress and, when the metric falls below a user-specified tolerance, the iterations stop and convergence is declared [see [Publications [2][8]] for a detailed discussion].

The SmallK library provides implementations of several different NMF algorithms. These algorithms are:

1. Multiplicative Updating (NMF-MU)
2. Hierarchical Alternating Least Squares (NMF-HALS)
3. Block Principal Pivoting (NMF-BPP)
4. Rank2 Specialization (NMF-RANK2)

SmallK also provides implementations of hierarchical and flat clustering. These routines are:

1. Hierarchical Clustering via NMF-RANK2
2. Flat Clustering via NMF-RANK2
3. Flat Clustering via NMF-BPP or NMF-HALS

The suite of SmallK implementations of NMF algorithms are suitable in many applications such as image processing, interactive visual analytics, speckle removal from SAR images, recommender systems, information fusion, outlier detection, chemometrics, and many more.

The SmallK library requires either MacOSX or Linux. A Windows version via a Vagrant installation is also available.

[8] http://smallk.github.io/publications/

2.4 Prerequisites

The following list is the software packages/libraries required to build the SmallK NMF library code:

- A modern, C++11-compliant compiler, such as g++ 4.9 or later
- Elemental[9], a high-performance library for dense, distributed linear algebra, which requires:
  - An MPI installation, such as OpenMPI[10] or mpich[11]
  - BLAS implementation, hopefully optimized/tuned for the local system
  - libFLAME[12]: a high-performance library for dense numerical linear algebra
  - CMake
- Python 2.7 (optional), including the following libraries (required to build the Python interface to SmallK, which is optional):
  - numpy
  - scipy
  - cython

Elemental can make use of OpenMP or mpich parallelization if available, which is generally advantageous for large problems. The SmallK code is also internally parallelized to take full advantage of multiple CPU cores for maximum performance. SmallK does not currently support distributed computation, but this is planned for future updates.

The SmallK software supports the latest stable release of Elemental, version 0.85

Check the documentation links on this page for additional detailed instructions for installation of the SmallK library software and dependencies. If desired, please see also the installation instructions for Elemental[13].

[9] http://libelemental.org/
[10] http://www.open-mpi.org/software/ompi/v1.6/
[11] http://http://www.mpich.org/
[12] http://www.cs.utexas.edu/~flame/web/libFLAME.html
[13] http://libelemental.org/documentation/

CHAPTER

THREE

QUICKSTART - INSTALLATION

3.1 Vagrant Virtual Machine

Installing SmallK into a virtual machine (OSX, Linux, Windows) is intended for those who are not doing development and/or do not have a reason to do the full installation on Linux or OSX outlined in the sections to follow.

The complete stack of software dependencies for SmallK as well as SmallK itself can be rapidly set up and configured through use of Vagrant and VirtualBox and the files included in the repository. The Vagrant install has been tested on Linux Ubuntu 16.04, Mac OSX Sierra 10.12.6, and Windows 10.

Note that the `smallk/vagrant/bootstrap.sh` file can be modified to perform various tasks when provisioning the vagrant session. Consider customizing `bootstrap.sh` to set up a custom install of libsmallk as required.

To deploy the SmallK VM:

1. Install Vagrant[14] and VirtualBox[15].

Tip: Note: For Windows, ensure that you have a VirtualBox version >= 4.3.12. After installing Vagrant, you may need to log out and log back in to ensure that you can run vagrant commands in the command prompt.

Optional: `git clone` the smallk_data[16] repository so that it is parallel with the smallk repository. This is an alternate way to test the installation and begin to work with SmallK. This directory can be synced with a directory of the same name in the VM by adding or uncommenting the following line in `smallk/vagrant/Vagrantfile`:

```
config.vm.synced_folder "../../smallk_data", "/home/vagrant/smallk_data"
```

2. From within the `smallk/vagrant` directory, run:

```
vagrant up
```

This can take as long as an hour to build the VM, which will be based on a minimal Ubuntu 16.04 installation. The `smallk/vagrant/Vagrantfile` can be customized in many ways to change the specifications for the VM that is built. See more information here[17]. The default configuration provides the VM with 4 GB of memory and 3 CPUs. Increasing these allocations will improve the performance of the application. This can be done by modifying these lines in the Vagrantfile:

```
vb.memory = 4096
vb.cpus = 3
```

[14] http://www.vagrantup.com/downloads.html
[15] https://www.virtualbox.org/wiki/Downloads
[16] https://github.com/smallk/smallk_data
[17] http://docs.vagrantup.com/v2/

After `vagrant up` has completed, the SmallK and pysmallk libraries will have been built and tested. Additionally, the smallk_data directory, if cloned as in the optional step above, will have been synced into the VM. For more details regarding what is being built and executed while provisioning the VM, please inspect `smallk/vagrant/bootstrap.sh`.

3. Once the VM has been built, run:

```
vagrant ssh
```

Tip: Note: For Windows, you will need an ssh client in order to run the above command. This can be obtained via CygWin[18] MinGW[19], or Git[20]. If you would like to use PuTTY to connect to your virtual machine, follow these[21] instructions.

In case you need it, the username/password for the VM created will be vagrant/vagrant.

This will drop you into the command line of the VM that was just created, in a working directory at `/home/vagrant`. From there, you can navigate to `/home/vagrant/libsmallk-<version>`, (e.g., libsmallk-1.6.2), and run:

```
make check PYSMALLK=1 ELEMVER=0.85 DATA_DIR=../smallk_data
```

to verify your installation was successful.

4. To test the installation at the command line, run:

```
nmf
```

This will produce the help output for the nmf library function:

```
Usage: nmf
        --matrixfile <filename>  Filename of the matrix to be factored.
                                 Either CSV format for dense or MatrixMarket format␣
↪for sparse.
        --k <integer value>      The common dimension for factors W and H.
        [--algorithm  BPP]       NMF algorithms:
                                     MU:    multiplicative updating
                                     HALS:  hierarchical alternating least squares
                                     RANK2: rank2 with optimal active set selection
                                     BPP:   block principal pivoting
        [--stopping  PG_RATIO]   Stopping criterion:
                                     PG_RATIO: Ratio of projected gradients
                                     DELTA:    Change in relative F-norm of W
        [--tol  0.005]           Tolerance for the selected stopping criterion.
        [--tolcount  1]          Tolerance count; declare convergence after this many
                                 iterations with metric < tolerance; default is to
                                 declare convergence on the first such iteration.
        [--infile_W  (empty)]    Dense mxk matrix to initialize W; CSV file.
                                 If unspecified, W will be randomly initialized.
        [--infile_H  (empty)]    Dense kxn matrix to initialize H; CSV file.
                                 If unspecified, H will be randomly initialized.
        [--outfile_W  w.csv]     Filename for the W matrix result.
        [--outfile_H  h.csv]     Filename for the H matrix result.
        [--miniter  5]           Minimum number of iterations to perform.
        [--maxiter  5000]        Maximum number of iterations to perform.
```

[18] https://www.cygwin.com/
[19] http://sourceforge.net/projects/mingw/files/
[20] http://git-scm.com/downloads
[21] https://github.com/Varying-Vagrant-Vagrants/VVV/wiki/Connect-to-Your-Vagrant-Virtual-Machine-with-PuTTY

```
        [--outprecision  6]      Write results with this many digits of precision.
        [--maxthreads    3]      Upper limit to thread count.
        [--normalize  1]         Whether to normalize W and scale H.
                                     1 == yes, 0 == no
        [--verbose  1]           Whether to print updates to the screen.
                                     1 == print updates, 0 == silent
```

5. To test the installation of pysmallk, attempt to import numpy and pysmallk; numpy must be imported BEFORE pysmallk is imported. Running the following command from the command line should produce no output:

```
python -c "import numpy; import pysmallk"
```

If there is no import error, pysmallk was installed correctly and is globally available.

6. When you are ready to shut down the VM, run `exit` from within the vagrant machine, then run one of the following from the command line of your host machine (wherever `vagrant up` was executed):

Save the current running state:

```
vagrant suspend
```

Gracefully shut down the machine:

```
vagrant halt
```

Remove the VM from your machine (this will require rebuilding the VM to restart it):

```
vagrant destroy
```

If you want to work with the VM again, from any of the above states you can run:

```
vagrant up
```

again and the VM will be resumed or recreated.

3.2 Docker Instructions

Running SmallK in a Docker container is intended for those who would like a fast, simple install that keeps their environment unmodified, in exchange for a loss in runtime performance. The basic process is to first build the Docker image, then run the Docker container to execute the desired command.

1. Install Docker[22]. If you are new to Docker, it may be worth exploring a quick introduction[23], or at least a cheat-sheet[24]. There are platform specific[25] installation, configuiration, and execution instructions for Mac, Windows, and Linux. The following instructions were tested on Ubuntu 16.04 with Docker version 17.06.0-ce.

2. Build the smallk Docker image.

First, make sure you have all submodules and their own submodules. From within the root of the smallk directory, run:

```
git submodule update --init --recursive
```

[22] https://docs.docker.com/engine/installation/
[23] https://docs.docker.com/get-started/
[24] https://github.com/wsargent/docker-cheat-sheet
[25] https://docs.docker.com/manuals/

Now we can build the image. In the same (project root) directory, run this:

```
docker build -t smallk .
```

This will download all dependencies from the Ubuntu repositories, PyPI, GitHub, etc. Everything will be built including smallk itself. You will end up with a Docker image tagged "smallk". At the end of the build process you should see the following:

```
Step 40/40 : CMD /bin/bash
 ---> Running in 3fdb5e73afdc
 ---> f8afa9f6a532
Removing intermediate container 3fdb5e73afdc
Successfully built f8afa9f6a532
Successfully tagged smallk:latest
```

This can take as long as an hour to build the image, which is based on a minimal Ubuntu 16.04 installation. The `smallk/Dockerfile` can be customized in many ways to change the specifications for the image that is built.

3. Run the Docker container.

The Docker container may be executed from any directory. Regardless of where you run it, you will need a volume for any input/output data. As an example, you may run the built-in PySmallk tests. The instructions below assume that your work directory is named `/home/ubuntu`. Replace it with the appropriate name. (The Docker daemon requires an absolute path for the local volume reference.):

```
cd /home/ubuntu
git clone https://github.com/smallk/smallk_data.git smallk_data
docker run --volume /home/ubuntu/smallk_data:/data smallk make check PYSMALLK=1␣
↪ELEMVER=0.85 DATA_DIR=/data
```

Here is a breakdown of that Docker command to explain each part:

- `docker run`: Run a new container from an image
  - `--volume`: Add a volume (persistent storage area) to the container
    - `/home/ubuntu/smallk_data`: Local absolute path that will be exposed within the running container
    - `/data`: Internal path to use within the container
  - `smallk`: Image tag from which to spawn the new container
  - `make check PYSMALLK=1 ELEMVER=0.85`: Command to run within the container (run the smallk test suite)
    - `DATA_DIR=/data`: Tell the test suite where the local data is stored (from the perspective of the container)

If your execution of the PySmallk tests is successful, you should see a lot of output, ending with the following lines:

```
assignment file test passed
***** PysmallK: All tests passed. *****
```

CHAPTER

FOUR

QUICKSTART - SMALLK API

4.1 Introduction

This document describes how to use the SmallK library to perform nonnegative matrix factorization (NMF), hierarchical clustering, and flat clustering. It is assumed that the library has been installed properly, that all tests have passed, and that the user has created the `SMALLK_INSTALL_DIR` environment variable as described in the documentation. SmallK provides a very simple interface to NMF and clustering algorithms. Examples of how to use this interface are described in this document. The SmallK distribution also provides a suite of command-line tools for NMF and clustering, suitable for advanced users.

4.2 C++ Project Setup

The SmallK distribution includes an `examples` folder containing two files: `smallk_examples.cpp` and a `Makefile`. To build the example CPP file, open a terminal window, `cd` to the `smallk/examples` folder, and run the command `make`.

If the SmallK library has been installed properly and the smallk_data[26] repository has been cloned at the same directory level as the SmallK library repository, the project should build and the binary file bin/example will be created. To run the example, run this command from the smallk/examples folder:

```
./bin/example ../../smallk_data
```

Results will appear for the following algorithms:

```
Running NMF-BPP using k=32
Running NMF-HALS using k=16
Running NMF-RANK2 with W and H initializers
Repeating the previous run with tol = 1.0e-5
Running HierNMF2 with 5 clusters, JSON format
Running HierNMF2 with 10 clusters, 12 terms, XML format
Running HierNmf2 with 18 clusters, 8 terms, with flat
```

The output files will be written to the directory where the binary `example` is run. In the above, the outputs will be written to the `<SmallK dir>/examples`.

To experiment with the SmallK library, make a backup copy of `smallk_examples.cpp` as follows:

```
cp smallk_examples.cpp smallk_examples.cpp.bak
```

[26] https://github.com/smallk/smallk_data

The file `smallk_examples.cpp` can now be used for experimentation. The original file can be restored from the backup at the user's discretion.

Delete lines 61-255 from `smallk_examples.cpp` (everything between the opening and closing braces of the `try` block). New code will be added between these braces in the steps below.

All of the examples described in this document use a matrix derived from Reuters articles. This matrix will be referred to as the `Reuters` matrix. It is a sparse matrix with 12411 rows and 7984 columns.

The SmallK documentation contains complete descriptions of all SmallK functions mentioned in this guide.

4.3 Load a Matrix

Suppose you want to perform NMF or clustering on a matrix. The first action to take is to load the matrix into SmallK using the `LoadMatrix` function. This function accepts either dense matrices in CSV format or sparse matrices in MatrixMarket format. Since we want to perform NMF and clustering on the Reuters matrix, we need to supply the path to the Reuters matrix file (`reuters.mtx`) as an argument to LoadMatrix. This path has already been setup in the code; the appropriate string variable is called `filepath_matrix`. Enter the following line after the opening brace of the try block after line 61:

```
smallk::LoadMatrix(filepath_matrix);
```

Save the file and run the following commands, which should complete without error:

```
make clean
make
```

Once a matrix is loaded into SmallK it remains loaded until it is replaced with a new call to LoadMatrix. Thus, SmallK makes it easy to experiment with different factorization or clustering parameters, without having to reload a matrix each time.

4.4 Perform NMF on the Loaded Matrix

Having loaded the Reuters matrix, we can now run different NMF algorithms and factor the matrix in various ways. The SmallK code factors the loaded matrix (denoted by A) as $\boldsymbol{A} \cong \boldsymbol{WH}$, where A is mxn, W is mxk, and H is kxn. The NMF is a low-rank approximation where the value of k, the rank, is an input parameter to the factorization routines, and is generally much smaller than either m or n. Matrix A can be either sparse or dense; matrices W and H are always dense.

4.4.1 NMF-BPP

Let's use the default NMF-BPP algorithm to factor the 12411 x 7984 Reuters matrix into W and H with a k value of 32. Add the following lines to the code:

```
MsgBox("Running NMF-BPP using k=32");
smallk::Nmf(32);
```

Build the code as described above; then run it with this command:

```
./bin/example ../smallk_data
```

The MsgBox function prints the string supplied as argument to the screen; this function is purely for annotating the output. The Nmf function performs the factorization and generates two output files, `w.csv` and `h.csv`, which contain the matrix factors. The files are written to the current directory. SmallK can write these files to a specified output directory via the SetOutputDir function, but we will use the current directory for the examples in this guide.

4.4.2 NMF-HALS

Now suppose we want to repeat the factorization, this time using the NMF-HALS algorithm with a k value of 16. Since the BPP algorithm is the default, we need to explicitly specify the algorithm as an argument to the Nmf function. Add these lines to the code:

```
MsgBox("Running NMF-HALS using k=16")
smallk::Nmf(16, smallk::Algorithm::HALS);
```

Build and run the code again; you should observe that the code now performs two separate factorizations.

4.4.3 NMF Initialization

The SmallK library provides the capability to explicitly initialize the W and H factors. For the previous two examples, these matrices were randomly initialized, since no initializers were provided in the call to the Nmf function. The data directory contains initializer matrices for the W and H factors of the Reuters matrix, assuming that k has a value of 2. To illustrate the use of initializers, we will use the RANK2 algorithm to factor the Reuters matrix again, using a k-value of 2, but with explicit initializers. Add these lines to the code:

```
MsgBox("Running NMF-RANK2 with W and H initializers");
smallk::Nmf(2, smallk::Algorithm::RANK2, filepath_w, filepath_h);
```

Build and run the code again, and observe that the code performs three separate factorizations.

The string arguments *filepath_w* and *filepath_h* are configured to point to the W and H initializer matrices in the data directory. Note how these are supplied as the third and fourth arguments to Nmf. For general matrix initializers, the W initializer must be a fully-dense matrix, in CSV format, with dimensions mxk, and the H initializer must be a fully-dense matrix, in CSV format, with dimensions kxn.

The main purpose of using initializer matrices is to generate deterministic output, such as for testing, benchmarking, and performance studies. You will notice that if you run the code repeatedly, the first two factorizations, which use random initializers, generate results that vary slightly from run to run. The third factorization, which uses initializers, always generates the same output on successive runs.

Typically the use of initializers is not required.

4.5 Hierarchical Clustering

Now let's perform hierarchical clustering on the Reuters matrix. To do this, we must first load the dictionary (or vocabulary) file associated with the Reuters data (a file called `reuters_dictionary.txt`). A string variable containing the full path to this file is provided in the `filepath_dict` variable. Add the following line to the code to load the Reuters dictionary:

```
smallk::LoadDictionary(filepath_dict);
```

As with the matrix file, the dictionary file remains loaded until it is replaced by another call to LoadDictionary.

With the matrix file and the dictionary file both loaded, we can perform hierarchical clustering on the Reuters data. For the first attempt we will generate a factorization tree containing five clusters. The number of clusters is specified as an argument to the clustering function. Add these lines to the code:

```
MsgBox("Running HierNMF2 with 5 clusters, JSON format");
smallk::HierNmf2(5);
```

Build and run the code.

The hierarchical clustering function is called `HierNmf2`. In the call above it will generate five clusters and generate two output files. One file will be called `assignments_5.csv`, a CSV file containing the cluster labels. The first entry in the file is the label for the first column (document) of the matrix; the second entry is the label for the second column, etc. Any entries that contain -1 are outliers; these represent the documents that were not assigned to any cluster.

The other output file will be called `tree_5.json`, a JSON file containing the cluster information. This file contains sufficient information to unambiguously reconstruct the factorization tree. If you open the file and examine the contents you can see the top terms assigned to each node. Leaf nodes have -1 for their left and right child indices. From an examination of the keywords at the leaf nodes, it is evident that this collection of Reuters documents is concerned with financial topics.

4.6 Flat Clustering

For the final example, let's generate a flat clustering result in addition to the hierarchical clustering result. We will also increase the number of terms per node to 8 and the number of clusters to 18. Add the following lines to the code:

```
MsgBox("Running HierNmf2 with 18 clusters, 8 terms, with flat");
smallk::SetMaxTerms(8);
smallk::HierNmf2WithFlat(18);
```

Build and run the code.

The call to SetMaxTerms increases the number of top terms per node. The next line runs the hierarchical clustering algorithm and also generates a flat clustering result. This time, four output files are generated. They are:

1. `assignments_18.csv`: assignments from hierarchical clustering
2. `assignments_flat_18.csv`: assignments from flat clustering
3. `tree_18.json`, the hierarchical factorization tree
4. `clusters_18.json`, the flat clustering results

These examples demonstrate how easy it is to use SmallK for NMF and clustering. There are additional functions in the SmallK interface, described in the documentation, installation section, which allows users to set various parameters that affect the NMF-based algorithms of SmallK. The default values for all such parameters are very reasonable, and most users will likely not ever need to change these parameters.

The `smallk_examples.cpp` file and the associated makefile can be used as a starting point for your own NMF and clustering projects.

4.7 Disclaimer

This software is a work in progress. It will be updated throughout the course of the XDATA program with additional algorithms and examples. The distributed NMF factorization routine uses sequential algorithms, but it replaces the matrices and matrix operations with distributed versions. The GA Tech research group is working on proper distributed

NMF algorithms, and when such algorithms are available they will be added to the library. Thus the performance of the distributed code should be viewed as being the baseline for our future distributed NMF implementations.

4.8 Contact Info

For comments, questions, bug reports, suggestions, etc., contact:

Barry Drake
Research Scientist
Information and Communications Laboratory (ICL)
Information and Cyber Sciences Directorate (ICSD)
Georgia Tech Research Institute (GTRI)
75 5TH St. NW STE 900
ATLANTA, GA 30308-1018
barry.drake@gtri.gatech.edu

Stephen Lee-Urban
Research Scientist
Information and Communications Laboratory (ICL)
Information and Cyber Sciences Directorate (ICSD)
Georgia Tech Research Institute (GTRI)
75 5TH St. NW STE 900
ATLANTA, GA 30308-1018
stephen.lee-urban@gtri.gatech.edu

CHAPTER

FIVE

INSTALLATION INSTRUCTIONS

5.1 Prerequisites

- A modern C++ compiler that supports the C++11 standard, such as the latest release of the GNU or clang compilers
- Elemental[27], a high-performance library for dense, distributed linear algebra, which requires:
  - An MPI installation, such as OpenMPI[28] and mpich[29]
  - A BLAS implementation, preferably optimized/tuned for the local system
  - libFLAME[30]: a high-performance library for dense linear algebra
  - OpenMP[31] (optional, see below)
  - CMake
- Python 2.7, including the following libraries:
  - numpy
  - scipy
  - cython version 0.22

5.1.1 Elemental

Elemental can make use of MPI parallelization if available. This is generally advantageous for large problems. The SmallK code is also internally parallelized to take full advantage of multiple CPU cores for maximum performance. SmallK does not currently support distributed computation. However, future updates are planned that provide this capability. Please see the About[32] page for information regarding distributed versions of many of the algorithms within SmallK.

We **strongly** recommend that users install both the HybridRelease and PureRelease builds of Elemental[33]. OpenMP is enabled in the HybridRelease build and disabled in the PureRelease build. So why install both? For smaller problems the overhead of *MPI can actually cause code to run slower* than without it. Whereas for large problems MPI parallelization generally helps, but there is no clear transition point between where it helps and where it hurts. Thus, we encourage users to experiment with both builds to find the one that performs best for their typical problems.

[27] http://libelemental.org/
[28] http://www.open-mpi.org/software/ompi/v1.6/
[29] http://www.mpich.org/
[30] http://www.cs.utexas.edu/~flame/web/libFLAME.html
[31] http://openmp.org/wp/
[32] http://smallk.github.io/about/
[33] http://libelemental.org/

We also recommend that users clearly separate the different build types as well as the versions of Elemental on their systems. Elemental is under active development, and new releases can introduce changes to the API that are not backwards-compatible with previous releases. To minimize build problems and overall hassle, we recommend that Elemental be installed so that the different versions and build types are cleanly separated.

Thus, two versions of Elemental need to be built. One is a hybrid release build with OpenMP parallelization, and the other is the pure release build without OpenMP parallelization. A separate build folder will be created for each build. The build that uses internal OpenMP parallelization is called a `HybridRelease` build; the build that doesn't is called a `PureRelease` build. The debug build is called a `PureDebug` build. The HybridRelease build is best for large problems, where the problem size is large enough to overcome the OpenMP parallel overhead. The following is for the 0.84 version of elemental. Set the version to that specified in the README.html file. Note that the files will be installed in `/usr/local/elemental/[version]/[build type]`.

The SmallK software supports the latest stable release of Elemental, version 0.85 and above.

5.1.1.1 How to Install Elemental on MacOSX

On MacOSX we recommend using Homebrew[34] as the package manager. Homebrew does not require sudo privileges for package installation, unlike other package managers such as MacPorts. Thus the chances of corrupting vital system files are greatly reduced using Homebrew.

It is convenient to be able to view hidden files (like .file) in the MacOSX Finder. To do so run the following at the command line:

```
defaults write com.apple.finder AppleShowAllFiles -bool YES
```

To revert back to hiding hidden files, set the Boolean flag to NO:

```
defaults write com.apple.finder AppleShowAllFiles -bool NO
```

If you use Homebrew, ensure that your PATH is configured to search Homebrew's installation directory first. Homebrew's default installation location is `/usr/local/bin`, so that location needs to be first on your path. To check, run this command from a terminal window:

```
cat /etc/paths
```

We also recommend running the following commands on a daily basis to refresh your brewed installations:

```
brew update
brew upgrade
brew cleanup
brew doctor
```

This will maintain your Homebrew installed software and diagnose any issues with the installations.

If the first entry is not `/usr/local/bin`, you will need to edit the `/etc/paths` file. This is a system file, so first create a backup. Move the line `/usr/local/bin` so that it is on the first line of the file. Save the file, then close the terminal session and start a new terminal session so that the path changes will take effect.

5.1.1.1.1 OSX:Install the latest GNU compilers

Elemental and SmallK both require a modern C++ compiler compliant with the C++11 standard. We recommend that you install the latest stable version of the clang and GNU C++ compilers. To do this, first install the XCode command line tools with this command:

[34] http://mxcl.github.io/homebrew/

```
xcode-select --install
```

If this command produces an error, download and install XCode from the AppStore, then repeat the command. If that should still fail, install the command line tools from the XCode preferences menu. After the installation completes, run this command from a terminal window:

```
clang++ --version
```

You should see output similar to this:

```
Apple LLVM version 8.1.0 (clang-802.0.42)
Target: x86_64-apple-darwin16.7.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

The latest version of the GNU compiler at the time of writing is g++-7 (gcc 7.1.0), which is provided by the `gcc` homebrew package. In addition to the gcc package, homebrew also provides a gcc49 package from the homebrew/versions tap. If this alternative gcc49 package is installed on your system it will prevent homebrew from symlinking the gcc package correctly. We recommend uninstalling the gcc49 versioned package and just using the gcc package instead. The Fortran compiler provided with the gcc package will also be configured to properly build numpy, which is required for the python interface to SmallK.

If you need to uninstall the gcc49 package, run the following commands:

```
brew uninstall gcc49
brew cleanup
brew doctor
```

Then install the gcc package as follows:

```
brew install gcc
```

The Apple-provided gcc and g++ will not be overwritten by this installation. The new compilers will be installed into `/usr/local/bin` as gcc-7, g++-7, and gfortran-6. The Fortran compiler is needed for the installation of MPI and for building the python interface to SmallK.

5.1.1.1.2 OSX:Install MPI Tools

Install the latest version of mpich[35] with Homebrew as follows:

```
brew install mpich
```

We recommend installing mpich rather than openMPI due to some superior features of mpich (prior versions of Elemental use openMPI, which can be installed using Homebrew as well). Also, Elemental 0.85 (discussed below) now uses mpich. Please see some discussion regarding openMPI vs mpich at: http://stackoverflow.com/questions/2427399/mpich-vs-openmpi

5.1.1.1.3 OSX:Install libFlame

Next we detail the installation of the high performance numerical library libflame. The library can be gotten from the libflame git repository on github.

[35] http://www.mpich.org/

It's important to perform the git clone into a subdirectory NOT called `flame` since this can cause name conflicts with the installation. Typically, a git clone is performed into a directory called `libflame`. However, other directory names will work as well. **Please do not use the directory name 'flame'**.

To obtain the latest version of the FLAME library, clone the FLAME git repository with this command:

```
git clone https://github.com/flame/libflame.git
```

Run the configure script in the top-level FLAME directory as follows (assuming the install path is `/usr/local/flame`):

```
./configure --prefix=/usr/local/flame --with-cc=/usr/local/bin/gcc-6 --with-ranlib=/
↪usr/local/bin/gcc-ranlib-6
```

A complete list of configuration options can be obtained by running `./configure --help`.

After the configuration process completes, build the FLAME library as follows:

```
make -j4
```

The `-j4` option tells Make to use four processes to perform the build. This number can be increased if you have a more capable system. Libflame will be installed with the following command:

```
make install
```

The FLAME library is now installed.

5.1.1.1.4 OSX:Install Elemental

Here is a recommended installation scheme for Elemental:

Choose a directory for the root of the Elemental installation. For example, this may be:

```
/usr/local/elemental
```

Download one of the SmallK-supported releases of Elemental, unzip and untar the distribution, and cd to the top-level directory of the unzipped distribution. This directory will be denoted by UNZIP_DIR in the following instructions.

We now recommend using Elemental 0.85 or later. Earlier versions will no longer be supported.

5.1.1.1.4.1 HybridRelease Build

From the Elemental-0.85 directory, run the following command to create a local build directory for the HybridRelease build:

```
mkdir build_hybrid
cd build_hybrid
```

Use the following CMake command for the HybridRelease build, substituting 0.85 for <VERSION_STRING>:

```
cmake -D CMAKE_INSTALL_PREFIX=/usr/local/elemental/<VERSION_STRING>/HybridRelease
-D CMAKE_BUILD_TYPE=HybridRelease
-D CMAKE_CXX_COMPILER=/usr/local/bin/g++-7
-D CMAKE_C_COMPILER=/usr/local/bin/gcc-7
-D CMAKE_Fortran_COMPILER=/usr/local/bin/gfortran-7
-D MATH_LIBS="/usr/local/flame/lib/libflame.a;-framework Accelerate"
-D ELEM_EXAMPLES=ON -D ELEM_TESTS=ON  ..
```

Note that we have installed g++-7 into `/usr/local/bin` and libFLAME into `/usr/local/flame`. Alter these paths, if necessary, to match the installation location on your system.

Once the CMake configuration step completes, you can build Elemental from the generated Makefiles with the following command:

```
make -j4
```

The -j4 option tells Make to use four processes to perform the build. This number can be increased if you have a more capable system.

After the build completes, install elemental as follows:

```
make install
```

For Elemental version 0.85 and later, you need to setup your system to find the Elemental dynamic libraries. Method 2 below is preferred:

1. If your Mac OSX is earlier than Sierra, then, in your startup script (`~/.bash_profile`) or in a terminal window, enter the following command on a single line, replacing VERSION_STRING as above:

```
export DYLD_LIBRARY_PATH=
        $DYLD_LIBRARY_PATH:/usr/local/elemental/VERSION_STRING/HybridRelease/lib/
```

2. If your Mac OSX is Sierra or higher Apple's System Integrity Protection (SIP) will prevent using the `DYLD_LIBRARY_PATH` variable. We highly discourage disabling SIP as a workaround. Instead, in your startup script (`~/.bash_profile`) or in a terminal window, enter the following command on a single line, replacing `VERSION_STRING` as above:

```
ln -s /usr/local/elemental/<VERSION_STRING>/HybridRelease/lib/*.dylib* /usr/local/lib
```

This will symlink the required Elemental libraries.

5.1.1.1.4.2 PureRelease Build

Run these commands to create a build directory for the PureRelease build:

```
cd ..
mkdir build_pure
cd build_pure
```

Then repeat the CMake configuration process, this time with the following command for the PureRelease build:

```
cmake -D CMAKE_INSTALL_PREFIX=/usr/local/elemental/<VERSION_STRING>/PureRelease
-D CMAKE_BUILD_TYPE=PureRelease -D CMAKE_CXX_COMPILER=/usr/local/bin/g++-7
-D CMAKE_C_COMPILER=/usr/local/bin/gcc-7
-D CMAKE_Fortran_COMPILER=/usr/local/bin/gfortran-7
-D MATH_LIBS="/usr/local/flame/lib/libflame.a;-framework Accelerate"
-D ELEM_EXAMPLES=ON -D ELEM_TESTS=ON ..
```

Repeat the build commands and install this build of Elemental.

For Elemental version 0.85 and later, you need to setup your system to find the Elemental dynamic libraries. Method 2 below is preferred:

1. If your Mac OSX is earlier than Sierra, then, in your startup script (`~/.bash_profile`) or in a terminal window, enter the following command on a single line, replacing `VERSION_STRING` as above:

```
export DYLD_LIBRARY_PATH=
        $DYLD_LIBRARY_PATH:/usr/local/elemental/VERSION_STRING/HybridRelease/lib/
```

2. If your Mac OSX is Sierra or higher Apple's System Integrity Protection (SIP) will prevent using the `DYLD_LIBRARY_PATH` variable. We highly discourage disabling SIP as a workaround. Instead, in your startup script (`~/.bash_profile`) or in a terminal window, enter the following command on a single line, replacing `VERSION_STRING` as above:

```
ln -s /usr/local/elemental/<VERSION_STRING>/HybridRelease/lib/*.dylib* /usr/local/lib
```

This will symlink the required Elemental libraries.

The two builds of Elemental are now complete.

To test the installation, follow Elemental's test instructions[36] for the SVD test to verify that Elemental is working correctly.

5.1.1.2 How to Install Elemental on Linux

We strongly recommend using a package manager for your Linux distribution for installation and configuration of the required dependencies. We cannot provide specific installation commands for every variant of Linux, so we specify the high-level steps below. The following was tested on a system with Ubuntu 16.04 installed.

5.1.1.2.1 Linux:Install the latest GNU compilers

We recommend installation of the latest stable release of the GNU C++ compiler, which is g++-6 at the time of this writing.

Also, install the latest version of GNU Fortran, which is needed for the installation of the Message Passing Interface (MPI) tools.

5.1.1.2.2 Linux:Install MPI Tools

Elemental version 0.85 and higher uses mpich[37] for its MPI implementation.:

```
sudo apt-get update
sudo apt-get install mpich
```

This completes the installation of the MPI tools. It should also be noted that the Open MP implementation of the MPI tools could also be used for the following installations.

5.1.1.2.3 Linux:Install libFlame

Next we detail the installation of the high performance numerical library libflame. The library can be gotten from the libflame git repository on github.

[36] http://libelemental.org/documentation/0.85/build.html
[37] http://www.mpich.org/

It's important to perform the git clone into a subdirectory NOT called `flame` since this can cause name conflicts with the installation. We normally do a git clone into a directory called `libflame`. However, other directory names will work as well, but not `flame`.

To obtain the latest version of the FLAME library, clone the FLAME git repository with this command:

```
git clone https://github.com/flame/libflame.git
```

Run the configure script in the top-level FLAME folder as follows (assuming you want to install to `/usr/local/flame`; if not, change the prefix path):

```
./configure --prefix=/usr/local/flame --with-cc=/usr/local/bin/gcc-6 --with-ranlib=/
↪usr/local/bin/gcc-ranlib-6
```

A complete list of configuration options can be obtained by running:

```
./configure --help
```

Then build and install the code as follows:

```
make -j4
make install
```

This completes the installation of the FLAME library.

5.1.1.2.4 Linux:Install an accelerated BLAS library

It is essential to link Elemental with an accelerated BLAS library for maximum performance. Linking Elemental with a 'reference' BLAS implementation will cripple performance, since the reference implementations are designed for correctness not speed.

If you do not have an accelerated BLAS on your system, you can download and build OpenBLAS. Download, unzip, and untar the tarball (version 0.2.19 as of this writing) and cd into the top-level folder. Build OpenBLAS with this command, assuming you have a 64-bit system:

```
make BINARY=64 USE_OPENMP=1
```

Install with this command, assuming the installation directory is `/usr/local/openblas/0.2.19/`:

```
make PREFIX=/usr/local/openblas/0.2.19/ install
```

This completes the installation of OpenBLAS.

5.1.1.2.5 Linux:Install Elemental

Here is our suggested installation scheme for Elemental:

We strongly recommend that users install both the HybridRelease and PureRelease builds of Elemental. MPI tools are enabled in the HybridRelease build and disabled in the PureRelease build. So why install both? For smaller problems the overhead of MPI can actually cause code to run slower than without it. On the other hand, for large problems, MPI parallelization generally helps. However, there is no clear transition point between where it helps and where it hurts. Thus, we encourage users to experiment with both builds to find the one that performs best for their typical problems.

Another strong recommendation is that users clearly separate the different build types as well as the versions of Elemental on their systems. Elemental is under active development, and new releases can introduce changes to the

API that are not backwards compatible with previous releases. To minimize build problems and overall hassle, we recommend that Elemental be installed so that the different versions and build types are cleanly separated.

Choose a directory for the root of the Elemental installation. A good choice is:

```
/usr/local/elemental
```

Download one of the SmallK-supported releases of Elemental (see above), unzip and untar the distribution, and cd to the top-level folder of the unzipped distribution. This directory will be denoted by `UNZIP_DIR` in the following instructions.

Note that Elemental version 0.85 or later is the version currently supported; earlier versions are not supported. If an earlier version is needed for Linux, use the following instructions.

For the first step of the installation, for Elemental versions prior to 0.85, we need to fix a few problems with the CMake configuration files. Open the following file in a text editor:

```
UNZIP_DIR/cmake/tests/OpenMP.cmake
```

On the first line of the file, change:

```
if(HYBRID)
```

to this:

```
if(ELEM_HYBRID)
```

Next, open this file in a text editor:

```
UNZIP_DIR/cmake/tests/Math.cmake
```

Near the first line of the file, change:

```
if(PURE)
```

to this:

```
if(ELEM_PURE)
```

Save both files.

Run these commands to create the required directories for the build types:

```
mkdir build_hybrid
mkdir build_pure
```

5.1.1.2.5.1 HybridRelease build

From the `Elemental-<VERSION>` folder, run the following command to change to the local build folder for the HybridRelease build:

```
cd build_hybrid
```

For the first step of the installation, we need to fix a few problems with the CMake configuration files. Open the following file in a text editor:

```
Elemental-<VERSION>/cmake/tests/OpenMP.cmake
```

On the first line of the file, change:

```
if(HYBRID)
```

to this:

```
if(ELEM_HYBRID)
```

Next, open this file in a text editor:

```
Elemental-<version>/cmake/tests/Math.cmake
```

Near the first line of the file, change:

```
if(PURE)
```

to this:

```
if(ELEM_PURE)
```

Save both files.

Run the following command to create a local build folder for the HybridRelease build:

```
cd build_hybrid
```

Use the following CMake command for the HybridRelease build:

```
cmake -D CMAKE_INSTALL_PREFIX=/usr/local/elemental/<VERSION>/HybridRelease
-D CMAKE_BUILD_TYPE=HybridRelease -D CMAKE_CXX_COMPILER=/usr/local/bin/g++-6
-D CMAKE_C_COMPILER=/usr/local/bin/gcc-6
-D CMAKE_Fortran_COMPILER=/usr/local/bin/gfortran-6
-D MATH_LIBS="/usr/local/flame/lib/libflame.a;-L/usr/local/openblas/0.2.19/ -
↪lopenblas -lm"
-D ELEM_EXAMPLES=ON -D ELEM_TESTS=ON  ..
```

Note that we have installed g++-6 into `/usr/local/bin` and libFLAME into `/usr/local/flame`. Alter these paths, if necessary, to match the installation location on your system.

If this command does not work on your system, you may need to define the `BLAS_LIBS` and/or `GFORTRAN_LIB` config options.

Version 0.85 of Elemental has an error in one of its cmake files. The file is:

```
Elemental-0.85/cmake/tests/CXX.cmake
```

Modify the first line of this file from:

```
include(FindCXXFeatures)
```

to:

```
include_directories(FindCXXFeatures)
```

since FindCXXFeatures is now a directory. After this change, Elemental should Make without errors.

Once the CMake configuration step completes, you can build Elemental from the generated Makefiles with the following command:

```
make -j4
```

The -j4 option tells Make to use four processes to perform the build. This number can be increased if you have a more capable system.

After the build completes, install elemental as follows:

```
make install
```

After installing Elemental version 0.85, setup the system to find the Elemental shared library. Either in the startup script (`~/.bashrc`) or in a terminal window, enter the following command on a single line, replacing `VERSION_STRING` as above:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/elemental/VERSION_STRING/
↪HybridRelease/lib/
```

5.1.1.2.5.2 PureRelease build

After this, run these commands to create a build folder for the PureRelease build:

```
cd ..
cd build_pure
```

Then repeat the CMake configuration process, this time with the following command for the PureRelease build:

```
cmake -D CMAKE_INSTALL_PREFIX=/usr/local/elemental/0.84-p1/PureRelease
-D CMAKE_BUILD_TYPE=PureRelease -D CMAKE_CXX_COMPILER=/usr/local/bin/g++-6
-D CMAKE_C_COMPILER=/usr/local/bin/gcc-6
-D CMAKE_Fortran_COMPILER=/usr/local/bin/gfortran-6
-D MATH_LIBS="/usr/local/flame/lib/libflame.a;-L/usr/local/openblas/0.2.19/ -
↪lopenblas -lm"
-D ELEM_EXAMPLES=ON -D ELEM_TESTS=ON  ..
```

If this command does not work on your system, you may need to define the `BLAS_LIBS` and/or `GFORTRAN_LIB` config options.

Repeat the build commands and install this build of Elemental. Then, if you installed a version of Elemental **prior** to the 0.84 release, edit the `/usr/local/elemental/<version>/PureRelease/conf/ElemVars` file and replace the CXX line as indicated above.

Version 0.85 of Elemental has an error in one of its cmake files. The file is:

```
Elemental-0.85/cmake/tests/CXX.cmake
```

Modify the first line of this file from:

```
include(FindCXXFeatures)
```

to:

```
include_directories(FindCXXFeatures)
```

since FindCXXFeatures is now a directory. After this change, Elemental should Make without errors.

If Elemental version 0.85 or later was installed, setup the system to find the Elemental shared library for the PureRelease build. Enter the following command in a terminal window on a single line, replacing `VERSION_STRING` as above:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/elemental/VERSION_STRING/
↪PureRelease/lib/
```

Note: set this variable to point to either the HybridRelease or the PureRelease build of the Elemental shared library whenever you want to use SmallK.

This completes the two builds of Elemental.

To test the installation, follow Elemental's test instructions[38] for the SVD test to verify that Elemental is working correctly.

5.1.2 Installation of Python libraries

Note: the following section for installing the Python libraries can be skipped if not needed.

5.1.2.1 OSX:Install Python libraries

5.1.2.1.1 Install Python scientific packages

Assuming that you have used brew to install gcc, as indicated earlier, you can run the following commands to install the necessary libraries:

```
brew install python
brew install numpy
brew install scipy
```

To check your installation, run:

```
brew test numpy
```

IMPORTANT: Check to see that your numpy installation has correctly linked to the needed BLAS libraries.

Ensure that you are running the correct python:

```
which python
```

This should print out `/usr/local/bin/python`. Open a python terminal by typing `python` at the command line and run the following:

```
import numpy as np
np.__config__.show()
```

You should see something similar to the following:

```
lapack_opt_info:
extra_link_args = ['-Wl,-framework', '-Wl,Accelerate']
extra_compile_args = ['-msse3']
define_macros = [('NO_ATLAS_INFO', 3)]

blas_opt_info:
```

[38] http://libelemental.org/documentation/0.85/build.html

```
extra_link_args = ['-Wl,-framework', '-Wl,Accelerate']
extra_compile_args = ['-msse3', '-I/System/Library/Frameworks/vecLib.framework/Header
↪']
define_macros = [('NO_ATLAS_INFO', 3)]
```

If you are using OpenBLAS, you should see that indicated as well.

5.1.2.1.2 Install Cython: a Python interface to C/C++

First install the Python Package Index utility, pip. Many Python packages are configured to use this package manager, Cython being one.:

```
brew install pip
```

Only Cython 0.22 is supported at this time. To check which version is installed on your system use this commands:

```
$ python
>> import Cython
>> Cython.__version__
'0.22'
>>
```

To install Cython version 0.22 (if not already installed):

```
pip uninstall cython
pip install cython==0.22
```

Check the version of cython as above to ensure that Cython version 0.22 is installed.

5.1.2.2 Linux:Install Python libraries

The Python libraries can easily be installed via pip and apt-get with the following commands:

```
apt-get install pip
pip install numpy
apt-get install python-scipy
pip uninstall cython
pip install cython==0.22
```

This also ensures that cython version 0.22 is installed, which is the currently supported version. The Makefile assumes an installation path of `/usr/local/lib/python2.7/site-packages` for the compiled library file. If you are not using apt-get to install your packages, you will need to tell the Makefile where the appropriate site-packages directory is located on your system. Setting the `SITE_PACKAGES_DIR` command line variable when running make accomplishes this. If this doesn't work, an alternative way to set this up is to add a line to the `.bash_profile` file (always back up first):

```
export SITE_PACKAGES_DIR="<path to lib/python2.7>/site-packages/"
```

This allows for special installations of Python such as Continuum Analytics' Anaconda[39] distribution site-packages to be accessed.

[39] https://www.continuum.io/

5.2 Build and Installation of SmallK

5.2.1 Obtain the Source Code

The source code for the SmallK library can be downloaded from the SmallK repository[40] on github. Once downloaded uncompress the tar ball and follow the installation instructions below.

5.2.2 Build the SmallK library

After downloading and unpacking the code tarball cd into the top-level `libsmallk1_<version>` directory, where version is `MAJOR.MINOR.PATCH` (for example 1.6.2). The makefiles assume that you followed our suggested installation plan for Elemental. If this is NOT the case you will need to do one of the following:

1. Create an environment variable called `ELEMENTAL_INSTALL_DIR` which contains the path to the root folder of your Elemental installation
2. Define the variable `ELEMENTAL_INSTALL_DIR` on the make command line
3. Edit the SmallK makefile so that it can find your Elemental installation

Assuming that the default install locations are acceptable, build the SmallK code by running this command from the root directory of the distribution:

```
make all PYSMALLK=1 ELEMVER=0.85
```

or:

```
make all PYSMALLK=0 ELEMVER=0.85
```

This will build the SmallK and pysmallk (optional; see section [Installation of Python libraries]) below for setup of the Python libraries) libraries and several command-line applications. These are:

1. `libsmallk.a`, the SmallK library
2. `preprocess_tf`, a command-line application for processing and scoring term-frequency matrices
3. `matrixgen`, a command-line application for generating random matrices
4. `nmf`, a command-line application for NMF
5. `hierclust`, a command-line application for fast hierarchical clustering
6. `flatclust`, a command-line application for flat clustering via NMF
7. `pysmallk.so`, if PYSMALLK=1 (0: default), the Python-wrapped SmallK library, making SmallK available via Python

5.2.3 Install the SmallK library

To install the code, run this command to install to the default location, which is `/usr/local/smallk`:

```
make install PYSMALLK=1 ELEMVER=0.85
```

or:

[40] https://github.com/smallk/smallk.github.io/tree/master/code

```
make install PYSMALLK=0 ELEMVER=0.85
```

This will install the binary files listed above into the `/usr/local/smallk/bin` directory, which needs to be on your path to run the executables from anywhere on your system and avoid prepending with the entire path. To install the binary code to a different location, either create an environment variable called `SMALLK_INSTALL_DIR` and set it equal to the desired installation location prior to running the install command, or supply a prefix argument:

```
make prefix=/path/to/smallk  install
```

If `PYSMALLK=1`, this will install pysmallk.so into the site-packages directory associated with the Python binary, which is determined by `brew install python` as discussed above or wherever the python distribution is installed on the system, e.g., Continuum's Anaconda Python[41] distribution is installed in the user's home directory. To install the Python library to a different location, create an environment variable called `SITE_PACKAGES_DIR` and set it equal to the desired installation location prior to running the install command, or supply this as an argument for make:

```
make SITE_PACKAGES_DIR=/path/to/site-packages install
```

Or, as a last resort, you can edit the top-level SmallK makefile to conform to the installation scheme of your system. You may need root privileges to do the installation, depending on where you choose to install it.

Before testing the installation, the test code needs to access data. The data is located in a separate github repository so that when cloning the code, the large amount of data is not included. The data repository is located on github at smallk_data[42]:

5.2.4 Check the build and installation

To test the build, run this command with `DATA_DIR` set to wherever the SmallK data repository was cloned:

```
make check PYSMALLK=1 ELEMVER=0.85 DATA_DIR=../smallk_data
```

or:

```
make check PYSMALLK=0 ELEMVER=0.85 DATA_DIR=../smallk_data
```

This will run a series of tests, none of which should report a failure. Sample output from a run of these tests can be found in section SmallK Test Results[43].

Note: if you installed Elemental version 0.85, you will need to configure your system to find the Elemental shared library. See the Elemental installation instructions above for information on how to do this.

The command-line applications can be built individually by running the appropriate make command from the top-level SmallK directory. These commands are:

```
To build the smallk library only:                ``make libsmallk``
To build the preprocessor only:                  ``make preprocessor``
To build the matrix generator only:              ``make matrixgen``
To build the nmf only:                           ``make nmf``
To build hierclust only:                         ``make hierclust``
To build flatclust only:                         ``make flatclust``
To build pysmallk only:                          ``make pysmallk``
```

This completes the SmallK NMF library installation.

[41] https://www.continuum.io/
[42] https://github.com/smallk/smallk_data
[43] http://smallk.github.io/documentation/tests/#smalk_tests

5.3 Build and Installation of pysmallk shared library

Before building pysmallk, you must ensure that you have already built the standard SmallK library and applications: `libsmallk`, `preprocessor`, `matrixgen`, `hierclust`, and `flatclust`.

All C++ and python libraries and applications can be built simultaneously by setting the `PYSMALLK` command line variable:

```
make PYSMALLK=1
```

To build pysmallk individually from the pysmallk subdirectory (`<path to SmallK>/libsmallk-<version>/pysmallk`):

```
make pysmallk
```

To check the library installation:

```
make pysmallk_check DATA_DIR=../smallk_data
```

This will run a series of tests, none of which should report a failure.

To install the shared library in a globally accessible location, enable the `PYSMALLK` command line variable and, if needed, specify an `INSTALLATION_DIR`.

The Makefile assumes an installation path of `/usr/local/lib/python2.7/site-packages` for the compiled library file. If you are not using brew to install your packages, you will need to tell the Makefile where the appropriate site-packages directory is located on your system. Setting the `INSTALLATION_DIR` command line variable when running make accomplishes this. Also, make sure that there is not another site-packages directory in your PATH before the site-packages you intend to use since `make install` will copy pysmallk.so to `/usr/local/lib/python2.7/site-packages` by default. Other Python distributions will probably interfere with the pysmallk installation.:

```
make install PYSMALLK=1      INSTALLATION_DIR=/usr/local/lib/python2.7/site-packages/
```

To uninstall the libraries:

```
.. code-block:: none
```

make uninstall PYSMALLK=1 INSTALLATION_DIR=/usr/local/lib/python2.7/site-packages/

5.4 Matrix file formats

The SmallK software supports comma-separated value (CSV) files for dense matrices and Matrix Market[44] files for sparse matrices.

For example, the 5x3 dense matrix:

```
42      47      52
43      48      53
44      49      54
45      50      55
46      51      56
```

would be stored in a CSV file as follows:

[44] http://math.nist.gov/MatrixMarket/formats.html

```
42,47,52
43,48,53
44,49,54
45,50,55
46,51,56
```

The matrix is loaded exactly as it appears in the file. **Internally, SmallK stores dense matrices in column-major order**. Sparse matrices are stored in **compressed column format**.

5.5 Disclaimer

This software is a work in progress. It will be updated throughout the course of the XDATA program with additional algorithms and examples. The distributed NMF factorization routine uses sequential algorithms, but it replaces the matrices and matrix operations with distributed versions. The GA Tech research group is working on proper distributed NMF algorithms, and when such algorithms are available they will be added to the library. Thus, the performance of the distributed code should be viewed as being the baseline for our future distributed NMF implementations.

5.6 Contact Info

For comments, questions, bug reports, suggestions, etc., contact:

Barry Drake
Research Scientist
Information and Communications Laboratory (ICL)
Information and Cyber Sciences Directorate (ICSD)
Georgia Tech Research Institute (GTRI)
75 5TH St. NW STE 900
ATLANTA, GA 30308-1018
barry.drake@gtri.gatech.edu

Stephen Lee-Urban
Research Scientist
Information and Communications Laboratory (ICL)
Information and Cyber Sciences Directorate (ICSD)
Georgia Tech Research Institute (GTRI)
75 5TH St. NW STE 900
ATLANTA, GA 30308-1018
stephen.lee-urban@gtri.gatech.edu

CHAPTER

SIX

COMMAND LINE TOOLS

6.1 Introduction

The SmallK library provides a number of algorithm implementations for low rank approximation of a matrix. These can be used for performing various data analytics tasks such as topic modeling, clustering, and dimension reduction. This section will provide more in-depth description of the tools available with examples that can be expanded/modified for other application domains.

Before diving into the various tools, it will be helpful to set up the command line environment to easily run the various executables that comprise the SmallK library. First the command line needs to know where to find the executable files to run the tools. Since while installing SmallK `make_install` was run, the executables are located in `/usr/local/smallk/bin`. Thus, this should be added to the `$PATH` system variable or added to the environment. The following command line performs the task of modifying the path avoiding the need to cd into directories were the tools are located:

```
export PATH=/usr/local/smallk/bin:$PATH
```

This allows the tools to be executed from any directory.

A subset of these tools are also available from the pysmallk library: smallkapi (mirrors the NMF command line application), matrixgen, preprocessor, flatclust, and hierclust. The command line arguments are the same as those documented below. These tools are available within the `/pysmallk/tests/ directory` and can be executed as follows:

```
[python binary] [tool].py [command line arguments]
```

For example:

```
python preprocessor.py --indir smallk_data
```

6.2 Preprocessor

6.2.1 Overview

The preprocessor prunes rows and columns from term-frequency matrices, attempting to generate a result matrix that is more suitable for clustering. It also computes tf-idf weights for the remaining entries. Therefore the input matrix consists of nonnegative integers, and the output matrix consists of floating point numbers between 0.0 and 1.0. The MatrixMarket file format is used for the input and output matrices.

Rows (terms) are pruned if a given term appears in fewer than `DOCS_PER_TERM` documents. The value of `DOCS_PER_TERM` is a command-line parameter with a default value of 3. For a term-frequency input matrix, in

which the matrix elements represent occurrence counts for the terms, this parameter actually specifies the minimum row sum for each term. Any rows whose rowsums are less than this value will be pruned.

Columns (docs) are pruned if a given document contains fewer than `TERMS_PER_DOC` terms. The value of `TERMS_PER_DOC` is a command-line parameter with a default value of 5.

Whenever columns (documents) are pruned the preprocessor checks the remaining columns for uniqueness. Any duplicate columns are identified and a representative column is chosen as the survivor. The code always selects the column with the largest column index in such groups as the survivor. The preprocessor continues to prune rows and columns until it finds no further candidates for pruning. It then computes new tf-idf scores for the resulting entries and writes out the result matrix in MatrixMarket format.

If the preprocessor should prune all rows or columns, it writes an error message to the screen and terminates without generating any output.

6.2.2 Input Files

The preprocessor requires three input files: a matrix file, a dictionary file, and a document file. The matrix file contains a sparse matrix in MatrixMarket format (.mtx). This is a term-frequency matrix, and all entries should be positive integers. The preprocessor can also read in matrices containing floating-point inputs, but only if `boolean mode` is enabled; this will be described below. The preprocessor does not support dense matrices, since the typical matrices encountered in topic modeling problems are extremely sparse, with occupancies generally less than 1%.

The second file required by the preprocessor is a `dictionary file`. This is a simple ASCII text file containing one entry per line. Entries represent keywords, bigrams, or other general text strings the user is interested in. Each line of the file is treated as a `keyword`, so multi-word keywords are supported as well. The smallk/data folder contains a sample dictionary file called `dictionary.txt`. The first few entries are:

```
triumph
dey
canada
finger
circuit
...
```

The third file required by the preprocessor is a `documents file`. This is another simple ASCII text file containing one entry per line. Entries represent document names or other unique identifiers. The smallk/data folder also contains a sample documents file called `documents.txt`. The first few entries of this file are:

```
52828-11101.txt
51820-10202.txt
104595-959.txt
60259-3040.txt
...
```

These are the unique document identifiers for the user who generated the file. Your identifiers will likely have a different format.

Finally, the preprocessor requires these files to have the following names: matrix.mtx, dictionary.txt, and documents.txt. The input folder containing these files can be specified on the command line (described below). The output of the preprocessor is a new set of files called `reduced_matrix.mtx`, `reduced_dictionary.txt`, and `reduced_documents.txt`.

6.2.3 Command Line Options

The preprocessor binary is called `preprocess_tf`, to emphasize the fact that it operates on term-frequency matrices. If the binary is run with no arguments, it prints out the following information:

```
preprocess_tf
     --indir <path>
     [--outdir (defaults to current directory)]
     [--docs_per_term 3]
     [--terms_per_doc 5]
     [--maxiter 1000]
     [--precision 4]
     [--boolean_mode 0]
```

Only the first parameter, `--indir`, is required. All remaining params are optional and have the default values indicated.

The meanings of the various options are as follows:

1. `--indir`: path to the folder containing the files `matrix.mtx`, `dictionary.txt`, and `documents.txt`; may be in small_data for example
2. `--outdir`: path to the folder to into which results should be written
3. `--docs_per_term`: any rows whose entries sum to less than this value will be pruned
4. `--terms_per_doc`: any columns whose entries sum to less than this value will be pruned
5. `--maxiter`: perform no more than this many iterations
6. `--precision`: the number of digits of precision with which to write the output matrix
7. `--boolean_mode`: all nonzero matrix elements will be treated as if they had the value 1.0. In other words, the preprocessor will ignore the actual frequency counts and treat all nonzero entries as if they were 1.0.

6.2.4 Sample Runs

Here is a sample run of the preprocessor using the data provided in the smallk distribution. This run was performed from the top-level smallk folder after building the code:

```
preprocessor/bin/preprocess_tf --indir data

  Command line options:

              indir: data/
             outdir: current directory
      docs_per_term: 3
      terms_per_doc: 5
           max_iter: 1000
          precision: 4
       boolean_mode: 0

     Loading input matrix data/matrix.mtx
             Input file load time: 1.176s.

     Starting iterations...
             [1] height: 39771, width: 11237, nonzeros: 877453
     Iterations finished.
     New height: 39727
```

```
    New width: 11237
    New nonzero count: 877374
    Processing time: 0.074s.

    Writing output matrix reduced_matrix.mtx
    Output file write time: 2.424s.
    Writing dictionary file reduced_dictionary.txt
    Writing documents file reduced_documents.txt
    Dictionary + documents write time: 0.08s.
```

6.3 Matrixgen

6.3.1 Overview

The matrix generator application is a simple tool for generating simple matrices. These matrices can be loaded by the NMF and clustering tools for various testing scenarios. Use of the matrix generator is entirely optional.

6.3.2 Command Line Options

Running the matrixgen binary with no options generates the following output:

```
matrixgen

Usage: matrixgen
        --height <number of rows>
        --width  <number of cols>
        --filename <path>
        [--type  UNIFORM]  UNIFORM:     matrix with uniformly-distributed random␣
↪entries
                   DENSE_DIAG:  dense diagonal matrix with uniform random entries
                   SPARSE_DIAG: sparse diagonal matrix with uniform random entries
                   IDENTITY:    identity matrix
                   ONES:        matrix of all ones
                   ZEROS:       matrix of all zeros
                   SPARSE:      sparse matrix with uniform random entries
                                specify 'nz_per_col' to control occupancy

        [--rng_center  0.5]   center of random numbers
        [--rng_radius  0.5]   radius of random numbers
        [--precision   6]     digits of precision
        [--nz_per_col  1]     (SPARSE only) nonzeros per column
```

The `--height`, `--width`, and `--filename` options are required. All others are optional and have the default values indicated.

The meanings of the various options are as follows:

1. `--height`: number of rows in the generated matrix
2. `--width`: number of columns in the generated matrix
3. `--filename`: name of the output file
4. `--type`: the type of matrix to be generated; the default is a uniformly-distributed random matrix
5. `--rng_center`: random number distribution will be centered on this value

6. `--rng_radius`: random numbers will span this distance to either side of the center value
7. `--precision`: the number of digits of precision with which to write the output matrix
8. `--nz_per_col`: number of nonzero entries per sparse matrix column; valid only for SPARSE type

6.3.3 Sample Runs

Suppose we want to generate a matrix of uniformly-distributed random numbers. The matrix should have a height of 100 and a width of 16, and should be written to a file called `w_init.csv`. Use the matrix generator as follows:

```
matrixgen --height 100 --width 16 --filename w_init.csv
```

6.4 Nonnegative Matrix Factorization (NMF)

6.4.1 Overview

The NMF command line application performs nonnegative matrix factorization on dense or sparse matrices. If the input matrix is denoted by A, nonnegative matrix factors Wand H are computed such that $\boldsymbol{A} \cong \boldsymbol{W}\boldsymbol{H}$.

Matrix $\boldsymbol{A}$ can be either dense or sparse; matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ are always dense. Matrix $\boldsymbol{A}$ has m rows and n columns; matrix $\boldsymbol{W}$ has m rows and k columns; matrix $\boldsymbol{H}$ has k rows and n columns. Parameter k is a positive integer and is typically much less than either m or n.

6.4.2 Command Line Options

Running the nmf application with no command line parameters will cause the application to display all params that it supports. These are:

```
Usage: nmf
--matrixfile <filename>  Filename of the matrix to be factored.
                         Either CSV format for dense or MatrixMarket format for␣
↪sparse.
--k <integer value>      Inner dimension for factors W and H.
[--algorithm  BPP]       NMF algorithms:
                             MU:    multiplicative updating
                             HALS:  hierarchical alternating least squares
                             RANK2: rank2 with optimal active set selection
                             BPP:   block principal pivoting
[--stopping  PG_RATIO]   Stopping criterion:
                             PG_RATIO: Ratio of projected gradients
                             DELTA:    Change in relative F-norm of W
[--tol  0.005]           Tolerance for the selected stopping criterion.
[--tolcount  1]          Tolerance count; declare convergence after this many
                         iterations with metric < tolerance; default is to
                         declare convergence on the first such iteration.
[--infile_W  (empty)]    Dense mxk matrix to initialize W; CSV file.
                         If unspecified, W will be randomly initialized.
[--infile_H  (empty)]    Dense kxn matrix to initialize H; CSV file.
                         If unspecified, H will be randomly initialized.
[--outfile_W  w.csv]     Filename for the W matrix result.
[--outfile_H  h.csv]     Filename for the H matrix result.
[--miniter  5]           Minimum number of iterations to perform.
[--maxiter  5000]        Maximum number of iterations to perform.
```

```
[--outprecision  6]      Write results with this many digits of precision.
[--maxthreads    8]      Upper limit to thread count.
[--normalize  1]         Whether to normalize W and scale H.
                             1 == yes, 0 == no
[--verbose  1]           Whether to print updates to the screen.
                             1 == print updates, 0 == silent
```

The –matrixfile and –k options are required; all others are optional and have the default values indicated. The meanings of the various options are as follows:

1. `--matrixfile`: Filename of the matrix to be factored. CSV files are supported for dense matrices and MTX files for sparse matrices.
2. `--k`: the width of the W matrix (inner dimension of the matrix factors)
3. `--algorithm`: identifier for the factorization algorithm
4. `--stopping`: the method used to terminate the iterations; use PG_RATIO unless you have a specific reason not to
5. `--tol`: tolerance value used to terminate iterations; when the progress metric falls below this value iterations will stop; typical values are in the 1.0e-3 or 1.0e-4 range
6. `--tolcount`: a positive integer representing the number of successive iterations for which the progress metric must have a value <= tolerance; default is 1, which means the iterations will terminate on the first iteration with: `progress_metric <= tolerance`
7. `--infile_W`: CSV file containing the mxk initial values for matrix W; if omitted, W is randomly initialized
8. `--infile_H`: CSV file containing the kxn initial values for matrix H; if omitted, H is randomly initialized
9. `--outfile_W`: filename for the computed W factor; default is w.csv
10. `--outfile_H`: filename for the computed H factor; default is h.csv
11. `--miniter`: the minimum number of iterations to perform before checking progress; for smaller tolerance values, you may want to increase this number to avoid needless progress checks
12. `--maxiter`: the maximum number of iterations to perform
13. `--outprecision`: matrices W and H will be written to disk using this many digits of precision
14. `--maxthreads`: the maximum number of threads to use; the default is to use as many threads as the hardware can support (your number may differ from that shown)
15. `--normalize`: whether to normalize the columns of the W matrix and correspondingly scale the rows of H after convergence
16. `--verbose`: whether to display updates to the screen as the iterations progress

6.4.3 Sample Runs

The smallk distribution utilizes another repository smallk_data[45] (clone this repository from github) with a matrix file `reuters.mtx`. This is a tf-idf weighted matrix derived from the popular Reuters data set used in machine learning experiments.

Suppose we want to factor the Reuters matrix using a k value of 8. We would do that as follows, assuming that we are in the top-level smallk folder after building the code and that the `smallk_data` repository was cloned into *data*:

[45] https://github.com/smallk/smallk_data

```
nmf/bin/nmf --matrixfile data/reuters.mtx  --k 8
```

Note that if `make install` was run during installation and the $PATH variable or environment variable was set as above, this could also be called with:

```
usr/local/bin/nmf --matrixfile data/reuters.mtx  --k 8
```

If we want to instead use the HALS algorithm with k=16, a tolerance of 1.0e-4, and also perform 10 iterations prior to checking progress, we would use this command line:

```
nmf/bin/nmf --matrixfile data/reuters.mtx --k 16 --algorithm HALS --tol 1.0e-4 --
↪miniter 10
```

To repeat the previous experiment but with new names for the output files, we would do this:

```
nmf/bin/nmf --matrixfile data/reuters.mtx --k 16 --algorithm HALS --tol 1.0e-4
        --miniter 10 --outfile_W w_hals.csv -outfile_H h_hals.csv
```

6.5 Hierclust

6.5.1 Overview

First, we briefly describe the algorithm and the references section provides pointers to papers with detailed descriptions of the algorithms. NMF-RANK2 for hierarchical clustering generates a binary tree of items. We refer to a node in the binary tree and the items associated with the node interchangeably. This method begins by placing all data items in the root node. The number of leaf nodes to generate is specified (user input). The algorithm proceeds with the following steps, repeated until the maximum number of leaf nodes, `max_leaf_nodes`, is reached:

1. Pick the leaf node with the highest score (at the very beginning where only a root node is present, just pick the root node)
2. Apply NMF-RANK2 to the node selected in step 1, and generate two new leaf nodes
3. Compute a score for each of the two leaf nodes generated in step 2
4. Repeat until the desired number of leaf nodes has been generated

Step 2 implements the details of the node splitting into child nodes. Outlier detection plays a crucial role in hierarchical clustering to generate a tree with well-balanced and meaningful clusters. To implement this, we have two additional parameters in step 2: *trial_allowance* and *unbalanced*.

The parameter *trial_allowance* is the number of times that the program will try to split a node into two meaningful clusters. In each trial, the program will check if one of the two generated leaf nodes is an outlier set. If the outlier set is detected, the program will delete the items in the outlier set from the node being split and continue to the next trial. If all the trials are finished and the program still cannot find two meaningful clusters for this node, all the deleted items are "recycled" and placed into this node again, and this node will be labeled as a "permanent leaf node" that cannot be picked in step 1 in later iterations.

The parameter *unbalanced* is a threshold parameter to determine whether two generated leaf nodes are unbalanced. Suppose two potential leaf nodes L and R are generated from the selected node and L has fewer items than R. Let us denote the number of items in a node N as $|N|$. L and R are called *unbalanced* if

$|L| < unbalanced * (|L| + |R|)$

Note that if L and R are unbalanced, the potential node L with fewer items is not necessarily regarded as an outlier set. Please see the referenced paper for more details [3][46].

Internally, NMF-RANK2 is applied to each leaf node to compute the score in step 3. The computed result matrices W and H in step 3 are cached so that we can avoid duplicate work in step 2 in later iterations.

The score for each leaf node is based on a modified version of the NDCG (*Normalized Discounted Cumulative Gain*) measure, a common measure in the information retrieval community. A leaf node is associated with a "topic vector", and we can define "top terms" based on the topic vector. A leaf node will receive a high score if its top terms are a good combination of the top terms of its two potential children; otherwise it receives a low score.

The hierclust application generates two output files. One file contains the assignments of documents to clusters. This file contains one integer for each document (column) of the original matrix. The integers are the cluster labels for that cluster that the document was assigned to. If the document could not be assigned to a cluster, a -1 will be entered into the file, indicating that the document is an outlier.

The other output file contains information for each node in the factorization binary tree. The items in this file are:

1. `id`: a unique id for this node
2. `level`: the level in the tree at which this node appears; the root is at level 0, the children of the root are at level 1, etc.
3. `label`: the cluster label for this node (meaningful only for leaf nodes)
4. `parent_id`: the unique id of the parent of this node (the root node has parent_id == 0)
5. `parent_label`: the cluster label of the parent of this node
6. `left_child`: a Boolean value indicating whether this node is the left or right child of its parent
7. `left_child_label`: the cluster label of the left child of this node (leaf nodes have -1 for this value)
8. `right_child_label`: the cluster label of the right child of this node (leaf nodes have -1 for this value)
9. `doc_count`: the number of documents that this node represents
10. `top_terms`: the highest probability dictionary terms for this node

The node id values and the left or right child indicators can be used to unambiguously reconstruct the factorization tree.

6.5.2 Command Line Options

Running the hierclust application with no command line parameters will cause the application to display all params that it supports. These are:

```
Usage: hierclust/bin/hierclust
--matrixfile <filename>     Filename of the matrix to be factored.
                            Either CSV format for dense or MatrixMarket format for␣
↪sparse.
--dictfile <filename>       The name of the dictionary file.
--clusters <integer>        The number of clusters to generate.
[--initdir  (empty)]        Directory of initializers for all Rank2 factorizations.
                            If unspecified, random init will be used.
[--tol  0.0001]             Tolerance value for each factorization.
[--outdir  (empty)]         Output directory.  If unspecified, results will be
                            written to the current directory.
[--miniter  5]              Minimum number of iterations to perform.
[--maxiter  5000]           Maximum number of  iterations to perform.
```

[46] http://smallk.github.io/publications/

```
[--maxterms  5]             Number of terms per node.
[--maxthreads    8]         Upper limit to thread count.
[--unbalanced  0.1]         Threshold for determining leaf node imbalance.
[--trial_allowance  3]      Number of split attempts.
[--flat  0]                 Whether to generate a flat clustering result.
                                1 == yes, 0 == no
[--verbose  1]              Whether to print updates to the screen.
                                1 == yes, 0 == no
[--format  XML]             Format of the output file containing the tree.
                                XML: XML format
                                JSON: JavaScript Object Notation
[--treefile  tree_N.ext]    Name of the output file containing the tree.
                            N is the number of clusters for this run.
                            The string 'ext' depends on the desired format.
                            This filename is relative to the outdir.
[--assignfile assignments_N.csv]  Name of the file containing final assignments.
                                  N is the number of clusters for this run.
                                  This filename is relative to the outdir.
```

The `--matrixfile`, `--dictfile`, and `--clusters` options are required; all others are optional and have the default values indicated. The meanings of the various options are as follows:

1. `--matrixfile`: Filename of the matrix to be factored. CSV files are supported for dense matrices and MTX files for sparse matrices.
2. `--dictfile`: absolute or relative path to the dictionary file
3. `--clusters`: the number of leaf nodes (clusters) to generate
4. `--initdir`: Initializer matrices for W and H are loaded from the initdir directory. The matrices are assumed to have the names `Winit_1.csv`, `Hinit_1.csv`, `Winit_2.csv`, `Hinit_2.csv`, etc. It is up to the user to ensure that enough matrices are present in this dir to run the HierNMF2 code to completion. The number of matrices used is non-deterministic, so trial-and-error may be required to find a lower bound on the matrix count. This feature is used for testing (such as comparisons with Matlab), in which each factorization problem has to proceed from a known initializer. The W initializer matrices must be of shape m x 2, and the H initializer matrices must be of shape 2 x n.
5. `--tol`: tolerance value for each internal NMF-RANK2 factorization; the stopping criterion is the ratio of projected gradient method
6. `--outdir`: path to the folder into which to write the output files; if omitted results will be written to the current directory
7. `--miniter`: minimum number of iterations to perform before checking progress on each NMF-RANK2 factorization
8. `--maxiter`: the maximum number of iterations to perform on each NMF-RANK2 factorization
9. `--maxterms`: the number of dictionary keywords to include in each node
10. `--maxthreads`: the maximum number of threads to use; the default is to use as many threads as the hardware can support (your number may differ from that shown)
11. `--unbalanced`: threshold value for declaring leaf node imbalance (see explanation above)
12. `--trial_allowance`: maximum number of split attempts for any node (see explanation above)
13. `--flat`: whether to generate a flat clustering result in addition to the hierarchical clustering result
14. `--verbose`: whether to display updates to the screen as the iterations progress
15. `--format`: file format to use for the clustering results

16. `--treefile`: name of the output file for the factorization tree; uses the format specified by the format parameter
17. `--assignfile`: name of the output file for the cluster assignments

6.5.3 Sample Runs

The smallk distribution has available a `smallk_data` repository on github with a matrix file `reuters.mtx` and an associated dictionary file `reuters_dictionary.txt`. These files are derived from the popular Reuters data set used in machine learning experiments.

As above, it is assumed that the smallk_data[47] repository was cloned into `data` and that the commands can be run as below or from `/usr/local/bin`.

Suppose we want to perform hierarchical clustering on this data set and generate 10 leaf nodes. We would do that as follows, assuming that we are in the top-level smallk folder after building the code:

```
hierclust/bin/hierclust --matrixfile data/reuters.mtx  --dictfile data/reuters_
↪dictionary.txt --clusters 10
```

This will generate two result files in the current directory: `tree_10.xml` and `assignments_10.csv`.

If we want to instead generate 10 clusters, each with 8 terms, using JSON output format, we would use this command line:

```
hierclust/bin/hierclust --matrixfile data/reuters.mtx  --dictfile data/reuters_
↪dictionary.txt --clusters 10 --maxterms 8 --format JSON
```

Two files will be generated: `tree_10.json` and `assignments_10.csv`. The json file will have 8 keywords per node, whereas the `tree_10.xml` file will have only 5.

To generate a flat clustering result (in addition to the hierarchical clustering result), use this command line:

```
hierclust/bin/hierclust --matrixfile data/reuters.mtx  --dictfile data/reuters_
↪dictionary.txt --clusters 10 --maxterms 8 --format JSON --flat 1
```

Two additional files will be generated this time (along with `tree_10.json` and `assignments_10.csv`): `clusters_10.json`, which contains the flat clustering results, and `assignments_flat_10.csv`, which contains the flat clustering assignments.

6.6 Flatclust

6.6.1 Overview

The flatclust command line application factors the input matrix using either NMF-HALS or NMF-BPP and generates a flat clustering result. A flatclust run generating k clusters will generally run more slowly than a hierclust run, of the same number of clusters, with the –flat option enabled. The reason for this is that the hierclust application uses the NMF-RANK2 algorithm and always generates factor matrices with two rows or columns. The runtime of NMF scales superlinearly with k in this case, and thus runs fastest for the smallest k value.

The flatclust application generates two output files. The first file contains the assignments of documents to clusters and is interpreted identically to that of the hierclust application, with the exception that there are no outliers generated by flatclust.

[47] https://github.com/smallk/smallk_data

The second file contains the node information. This file is much simpler than that of the hierclust application since there is no factorization tree. The items for each node in this file are:

1. `id`: the unique id of this node
2. `doc_count`: the number of documents assigned to this node
3. `top_terms`: the highest probability dictionary terms assigned to this node

6.6.2 Command Line Options

Running the flatclust application with no command line parameters will cause the application to display all params that it supports. These are:

```
Usage: flatclust
--matrixfile <filename>      Filename of the matrix to be factored.
                             Either CSV format for dense or MatrixMarket format for␣
↪sparse.
--dictfile <filename>        The name of the dictionary file.
--clusters <integer>         The number of clusters to generate.
[--algorithm  BPP]           The NMF algorithm to use:
                                 HALS:  hierarchical alternating least squares
                                 RANK2: rank2 with optimal active set selection
                                        (for two clusters only)
                                 BPP:   block principal pivoting
[--infile_W  (empty)]        Dense matrix to initialize W, CSV file.
                             The matrix has m rows and 'clusters' columns.
                             If unspecified, W will be randomly initialized.
[--infile_H  (empty)]        Dense matrix to initialize H, CSV file.
                             The matrix has 'clusters' rows and n columns.
                             If unspecified, H will be randomly initialized.
[--tol  0.0001]              Tolerance value for the progress metric.
[--outdir  (empty)]          Output directory.  If unspecified, results will be
                             written to the current directory.
[--miniter  5]               Minimum number of iterations to perform.
[--maxiter  5000]            Maximum number of  iterations to perform.
[--maxterms  5]              Number of terms per node.
[--maxthreads  8]            Upper limit to thread count.
[--verbose  1]               Whether to print updates to the screen.
                                 1 == yes, 0 == no
[--format  XML]              Format of the output file containing the tree.
                                 XML: XML format
                                 JSON: JavaScript Object Notation
[--clustfile clusters_N.ext] Name of the output XML file containing the tree.
                             N is the number of clusters for this run.
                             The string 'ext' depends on the desired format.
                             This filename is relative to the outdir.
[--assignfile assignments_N.csv]  Name of the file containing final assignments.
                                  N is the number of clusters for this run.
                                  This filename is relative to the outdir.
```

The `--matrixfile`, `--dictfile`, and `--clusters` options are required; all others are optional and have the default values indicated. The meanings of the various options are as follows:

1. `--matrixfile`: Filename of the matrix to be factored. CSV files are supported for dense matrices and MTX (matrix market) files for sparse matrices.
2. `--dictfile`: absolute or relative path to the dictionary file

3. `--clusters`: the number of clusters to generate (equivalent to the NMF `k` value)
4. `--algorithm`: the factorization algorithm to use
5. `--infile_W`: CSV file containing the m x `clusters` initial values for matrix W; if omitted, W is randomly initialized
6. `--infile_H`: CSV file containing the `clusters` x n initial values for matrix H; if omitted, H is randomly initialized
7. `--tol`: tolerance value for the factorization; the stopping criterion is the ratio of projected gradient method
8. `--outdir`: path to the folder into which to write the output files; if omitted results will be written to the current directory
9. `--miniter`: minimum number of iterations to perform before checking progress
10. `--maxiter`: the maximum number of iterations to perform
11. `--maxterms`: the number of dictionary keywords to include in each node
12. `--maxthreads`: the maximum number of threads to use; the default is to use as many threads as the hardware can support (your number may differ from that shown)
13. `--verbose`: whether to display updates to the screen as the iterations progress
14. `--format`: file format to use for the clustering results
15. `--clustfile`: name of the output file for the nodes; uses the format specified by the format parameter
16. `--assignfile`: name of the output file for the cluster assignments

6.6.3 Sample Runs

The smallk distribution has available a `smallk_data` repository on github with a matrix file `reuters.mtx` and an associated dictionary file `reuters_dictionary.txt`. These files are derived from the popular Reuters data set used in machine learning experiments.

As above, it is assumed that the smallk_data[48] repository was cloned into `data` and that the commands can be run as below or from `/usr/local/bin`.

Suppose we want to perform flat clustering on this data set and generate 10 clusters. We would do that as follows, assuming that we are in the top-level smallk folder after building the code:

```
flatclust/bin/flatclust --matrixfile data/reuters.mtx  --dictfile data/reuters_
↪dictionary.txt --clusters 10
```

This will generate two result files in the current directory: `clusters_10.xml` and `assignments_10.csv`.

If we want to instead generate 10 clusters, each with 8 terms, using JSON output format, we would use this command line:

```
flatclust/bin/flatclust --matrixfile data/reuters.mtx  --dictfile data/reuters_
↪dictionary.txt --clusters 10 --maxterms 8 --format JSON
```

Two files will be generated: `clusters_10.json` and `assignments_10.csv`. The json file will have 8 keywords per node, whereas the `clusters_10.xml` file will have only 5.

[48] https://github.com/smallk/smallk_data

CHAPTER

SEVEN

SMALLK API (C++)

7.1 Examples of API Usage

In the examples folder you will find a file called `smallk_example.cpp`. This file contains several examples of how to use the SmallK library. Also included in the examples folder is a makefile that you can customize for your use. Note that the SmallK library must first be installed before the example project can be built.

As an example of how to use the sample project, assume the smallk software has been installed into `/usr/local/smallk`. Also assume that the user chose to create the recommended environment variable `SMALLK_INSTALL_DIR` that stores the location of the top-level install folder, i.e. the user's `.bashrc` file contains this statement:

```
export SMALLK_INSTALL_DIR=/usr/local/smallk
```

To build the smallk example project, open a terminal window and cd to the smallk/examples folder and run this command:

```
make
```

To run the example project, run this command:

```
./bin/example ../../smallk_data
```

Note: the output will be *similar* to the following not identical since some problems are randomly initialized:

```
    Smallk major version: 1
    Smallk minor version: 0
    Smallk patch level:   0
    Smallk version string: 1.0.0
    Loading matrix...

    ************************************************************
    *                                                          *
    *                Running NMF-BPP using k=32                *
    *                                                          *
    ************************************************************
    Initializing matrix W...
    Initializing matrix H...

            parameters:

         algorithm: Nonnegative Least Squares with Block Principal Pivoting
stopping criterion: Ratio of Projected Gradients
            height: 12411
```

```
             width: 7984
                 k: 32
           miniter: 5
           maxiter: 5000
               tol: 0.005
        matrixfile: ../data/reuters.mtx
        maxthreads: 8

    1:  progress metric:    (min_iter)
    2:  progress metric:    (min_iter)
    3:  progress metric:    (min_iter)
    4:  progress metric:    (min_iter)
    5:  progress metric:    (min_iter)
    6:  progress metric:    0.0747031
    7:  progress metric:    0.0597987
    8:  progress metric:    0.0462878
    9:  progress metric:    0.0362883
    10: progress metric:    0.030665
    11: progress metric:    0.0281802
    12: progress metric:    0.0267987
    13: progress metric:    0.0236731
    14: progress metric:    0.0220778
    15: progress metric:    0.0227083
    16: progress metric:    0.0244029
    17: progress metric:    0.0247552
    18: progress metric:    0.0220007
    19: progress metric:    0.0173831
    20: progress metric:    0.0137033

    Solution converged after 39 iterations.

    Elapsed wall clock time: 4.354 sec.

    Writing output files...

    ************************************************************
    *                                                          *
    *               Running NMF-HALS using k=16                *
    *                                                          *
    ************************************************************
    Initializing matrix W...
    Initializing matrix H...

             parameters:

         algorithm: HALS
stopping criterion: Ratio of Projected Gradients
            height: 12411
             width: 7984
                 k: 16
           miniter: 5
           maxiter: 5000
               tol: 0.005
        matrixfile: ../data/reuters.mtx
        maxthreads: 8

    1:  progress metric:    (min_iter)
    2:  progress metric:    (min_iter)
```

```
    3:  progress metric:    (min_iter)
    4:  progress metric:    (min_iter)
    5:  progress metric:    (min_iter)
    6:  progress metric:    0.710219
    7:  progress metric:    0.580951
    8:  progress metric:    0.471557
    9:  progress metric:    0.491855
    10: progress metric:    0.531999
    11: progress metric:    0.353302
    12: progress metric:    0.201634
    13: progress metric:    0.1584
    14: progress metric:    0.142572
    15: progress metric:    0.12588
    16: progress metric:    0.113239
    17: progress metric:    0.0976934
    18: progress metric:    0.0821207
    19: progress metric:    0.0746089
    20: progress metric:    0.0720616
    40: progress metric:    0.0252854
    60: progress metric:    0.0142085
    80: progress metric:    0.0153269

    Solution converged after 88 iterations.

    Elapsed wall clock time: 1.560 sec.

    Writing output files...

    ************************************************************
    *                                                          *
    *       Running NMF-RANK2 with W and H initializers        *
    *                                                          *
    ************************************************************
    Initializing matrix W...
    Initializing matrix H...

             parameters:

          algorithm: Rank 2
stopping criterion: Ratio of Projected Gradients
             height: 12411
              width: 7984
                  k: 2
            miniter: 5
            maxiter: 5000
                tol: 0.005
         matrixfile: ../data/reuters.mtx
         maxthreads: 8

    1:  progress metric:    (min_iter)
    2:  progress metric:    (min_iter)
    3:  progress metric:    (min_iter)
    4:  progress metric:    (min_iter)
    5:  progress metric:    (min_iter)
    6:  progress metric:    0.0374741
    7:  progress metric:    0.0252389
    8:  progress metric:    0.0169805
    9:  progress metric:    0.0113837
```

```
    10: progress metric:    0.00761077
    11: progress metric:    0.0050782
    12: progress metric:    0.00338569

    Solution converged after 12 iterations.

    Elapsed wall clock time: 0.028 sec.

    Writing output files...

    ************************************************************
    *                                                          *
    *       Repeating the previous run with tol = 1.0e-5       *
    *                                                          *
    ************************************************************
    Initializing matrix W...
    Initializing matrix H...

            parameters:

         algorithm: Rank 2
stopping criterion: Ratio of Projected Gradients
            height: 12411
             width: 7984
                 k: 2
           miniter: 5
           maxiter: 5000
               tol: 1e-05
        matrixfile: ../data/reuters.mtx
        maxthreads: 8

    1:  progress metric:    (min_iter)
    2:  progress metric:    (min_iter)
    3:  progress metric:    (min_iter)
    4:  progress metric:    (min_iter)
    5:  progress metric:    (min_iter)
    6:  progress metric:    0.0374741
    7:  progress metric:    0.0252389
    8:  progress metric:    0.0169805
    9:  progress metric:    0.0113837
    10: progress metric:    0.00761077
    11: progress metric:    0.0050782
    12: progress metric:    0.00338569
    13: progress metric:    0.00225761
    14: progress metric:    0.00150429
    15: progress metric:    0.00100167
    16: progress metric:    0.000666691
    17: progress metric:    0.000443654
    18: progress metric:    0.000295213
    19: progress metric:    0.000196411
    20: progress metric:    0.000130604

    Solution converged after 27 iterations.

    Elapsed wall clock time: 0.061 sec.

    Writing output files...
    Minimum value in W matrix: 0.
```

```
Maximum value in W matrix: 0.397027.


************************************************************
*                                                          *
*      Running HierNMF2 with 5 clusters, JSON format       *
*                                                          *
************************************************************
loading dictionary...
creating random W initializers...
creating random H initializers...

    parameters:

        height: 12411
         width: 7984
    matrixfile: ../data/reuters.mtx
      dictfile: ../data/reuters_dictionary.txt
           tol: 0.0001
       miniter: 5
       maxiter: 5000
      maxterms: 5
    maxthreads: 8
[1] [2] [3] [4]

Elapsed wall clock time: 391 ms.
9/9 factorizations converged.

Writing output files...

************************************************************
*                                                          *
* Running HierNMF2 with 10 clusters, 12 terms, XML format  *
*                                                          *
************************************************************
creating random W initializers...
creating random H initializers...

    parameters:

        height: 12411
         width: 7984
    matrixfile: ../data/reuters.mtx
      dictfile: ../data/reuters_dictionary.txt
           tol: 0.0001
       miniter: 5
       maxiter: 5000
      maxterms: 12
    maxthreads: 8
[1] [2] [3] [4] [5] [6] dropping 20 items ...
[7] [8] [9]

Elapsed wall clock time: 837 ms.
21/21 factorizations converged.

Writing output files...

************************************************************
```

```
*                                                            *
*  Running HierNmf2 with 18 clusters, 8 terms, with flat     *
*                                                            *
************************************************************
creating random W initializers...
creating random H initializers...

    parameters:

        height: 12411
         width: 7984
    matrixfile: ../data/reuters.mtx
      dictfile: ../data/reuters_dictionary.txt
           tol: 0.0001
       miniter: 5
       maxiter: 5000
      maxterms: 8
    maxthreads: 8
[1] [2] [3] [4] [5] [6] dropping 20 items ...
[7] [8] [9] dropping 25 items ...
[10] [11] [12] [13] [14] [15] [16] [17]

Running NNLS solver...
1:  progress metric:    1
2:  progress metric:    0.264152
3:  progress metric:    0.0760648
4:  progress metric:    0.0226758
5:  progress metric:    0.00743562
6:  progress metric:    0.00280826
7:  progress metric:    0.00103682
8:  progress metric:    0.000361738
9:  progress metric:    0.000133087
10: progress metric:    5.84849e-05

Elapsed wall clock time: 1.362 s.
40/40 factorizations converged.

Writing output files...
```

The output files are written to the default directory or the directory specified on the command line.

7.2 SmallK API

The SmallK API is an extremely simplistic API for basic NMF and clustering. Users who require more control over the factorization or clustering algorithms can instead run one of the command-line applications in the SmallK distribution.

The SmallK API is exposed by the file smallk.hpp, which can be found in this location:

```
SMALLK_INSTALL_DIR/include/smallk.hpp.
```

All API functions are contained within the smallk namespace.

An example of how to use the API can be found in the file `examples/smallk_example.cpp`.

The smallk library maintains a set of state variables that are used to control the Nmf and clustering routines. Once set, the state variables maintain their values until changed by an API function. For instance, one state variable represents the matrix to be factored (or used for clustering). The API provides a function to load this matrix; once loaded, it

can be repeatedly factored without the need for reloading. The state variables and their default values are documented below.

All computations with the smallk library are performed in double precision.

7.2.1 Enumerations

The SmallK API provides two enumerated types, one for the supported NMF algorithms and one for the clustering file output format. These are:

```
enum Algorithm
{
        MU,       // Multiplicative Updating, Lee & Seung
        BPP,      // Block Principal Pivoting, Kim and Park
        HALS,     // Hierarchical Alternating Least Squares, Cichocki & Pan
        RANK2     // Rank2, Kuang and Park
};
```

The default NMF algorithm is BPP. The Rank2 algorithm is optimized for two-column or two-row matrices and is the underlying factorization routine for the clustering code.

```
enum OutputFormat
{
        XML,  // Extensible Markup Language
        JSON  // JavaScript Object Notation
};
```

7.2.2 API functions

7.2.2.1 Initialization and cleanup

```
void Initialize(int& argc, char**& argv)
```

Call this function first, before all others in the API; initializes Elemental and the smallk library.

```
bool IsInitialized()
```

Returns true if the library has been initialized via a call to Initialize(), false otherwise.

Call this function last, after all others in the API; performs cleanup for Elemental and the smallk library:

```
void Finalize()
```

7.2.2.2 Versioning

```
unsigned int GetMajorVersion()
```

Returns the major release version number of the library as an unsigned integer.

```
unsigned int GetMinorVersion()
```

Returns the minor release version number of the library as an unsigned integer.

```
unsigned int GetPatchLevel()
```

Returns the patch version number of the library as an unsigned integer.

```
std::string GetVersionString()
```

Returns the version of the library as a string, formatted as major.minor.patch.

7.2.2.3 Common functions

```
unsigned int GetOutputPrecision()
```

Returns the floating point precision with which numerical output will be written (i.e., the computed W and H matrix factors from the Nmf routine). The default precision is six digits.

```
void SetOutputPrecision(const unsigned int num_digits)
```

Sets the floating point precision with which numerical output will be written. Input values should be within the range [1, precision(double)]. Any inputs outside of this range will be adjusted.

```
unsigned int GetMaxIter()
```

Returns the maximum number of iterations allowed for NMF computations. The default value is 5000.

```
void SetMaxIter(const unsigned int max_iterations = 5000)
```

Sets the maximum number of iterations allowed for NMF computations. The default of 5000 should be more than sufficient for most computations.

```
unsigned int GetMinIter()
```

Returns the minimum number of NMF iterations. The default value is 5.

```
void SetMinIter(const unsigned int min_iterations = 5)
```

Sets the minimum number of NMF iterations to perform before checking for convergence. The convergence and progress estimation routines are non-trivial calculations, so increasing this value may result in faster performance.

```
unsigned int GetMaxThreads()
```

Returns the maximum number of threads used for NMF or clustering computations. The default value is hardware-dependent, but is generally the maximum number allowed by the hardware.

```
void SetMaxThreads(const unsigned int max_threads);
```

Sets an upper limit to the number of threads used for NMF and clustering computations. Inputs that exceed the capabilities of the hardware will be adjusted. This function is provided for scaling and performance studies.

```
void Reset()
```

Resets all state variables to their default values.

```
void SeedRNG(const int seed)
```

Seeds the random number generator (RNG) within the smallk library. Normally this RNG is seeded from the system time whenever the library is initialized. The RNG is the `19937` Mersenne Twister implementation provided by the C++ standard library.

```
void LoadMatrix(const std::string& filepath)
```

Loads a matrix contained in the given file. The file must either be a comma-separated value (.CSV) file for a dense matrix, or a MatrixMarket-format file (.MTX) for a sparse matrix. If the matrix cannot be loaded the library throws a std::runtime_error exception.

```
bool IsMatrixLoaded()
```

Returns true if a matrix is currently loaded, false if not.

```
std::string GetOuputDir()
```

Returns a string indicating the directory into which output files will be written. The default is the current directory.

```
void SetOutputDir(const std::string& outdir)
```

Sets the directory into which output files should be written. The `outdir` argument can either be an absolute or relative path. The default is the current directory.

7.2.2.4 NMF functions

```
void Nmf(const unsigned int k,
        const Algorithm algorithm     = Algorithm::BPP,
        const std::string& initfile_w = std::string(""),
        const std::string& initfile_h = std::string(""))
```

This function factors the input matrix A of nonnegative elements into nonnegative factors such that: $\boldsymbol{A} \cong \boldsymbol{W}\boldsymbol{H}$. If a matrix is not currently loaded a std::logic_error exception will be thrown. The default algorithm is NMF-BPP; provide one of the enumerated algorithm values to use a different algorithm.

Where A is mxn, W is mxk, and H is kxn. The value of k a user defined argument, e.g., for clustering applications, k is the number of clusters.

Optional initializer matrices can be provided for the W and H factors via the `initfile_w` and `initfile_h` arguments. These files must contain fully dense matrices in .CSV format. The W matrix initializer must have dimension mxk, and the H matrix initializer must have dimension kxn. If the initializer matrices do not match these dimensions exactly a std::logic_error exception is thrown. If initializers are not provided, matrices W and H will be randomly initialized.

The computed factors W and H will be written to the output directory in the files `w.csv` and `h.csv`.

Exceptions will be thrown (either from Elemental or smallk) in case of error.

```
const double* LockedBufferW(unsigned int& ldim, unsigned int& height, unsigned int&␣
↪width)
```

This function returns a READONLY pointer to the buffer containing the W factor computed by the Nmf routine, along with buffer and matrix dimensions. The `ldim`, `height`, and `width` arguments are all out parameters. The buffer has a height of `ldim` and a width of `width`. The matrix W has the same width but a height of `height`, which may differ from ldim. The W matrix is stored in the buffer in column-major order. See the examples/smallk_example.cpp file for an illustration of how to use this function.

```
const double* LockedBufferH(unsigned int& ldim, unsigned int& height, unsigned int&␣
↪width)
```

Same as LockedBufferW, but for the H matrix.

```
double GetNmfTolerance()
```

Returns the tolerance value used to determine NMF convergence. The default value is 0.005.

```
void SetNmfTolerance(const double tol=0.005)
```

Sets the tolerance value used to determine NMF convergence. The NMF algorithms are iterative, and at each iteration a progress metric is computed and compared with the tolerance value. When the metric falls below the tolerance value the iterations stop and convergence is declared. The tolerance value should satisfy 0.0 < tolerance < 1.0. Any inputs outside this range will cause a std::logic_error exception to be thrown. Clustering Functions

```
void LoadDictionary(const std::string& filepath)
```

Loads the dictionary used for clustering. The dictionary is an ASCII file of text strings as described in the preprocessor input files section below. If the dictionary file cannot be loaded a std::runtime_error exception is thrown.

```
unsigned int GetMaxTerms()
```

Returns the number of highest-probability dictionary terms to store per cluster. The default value is 5.

```
void SetMaxTerms(const unsigned int max_terms = 5)
```

Sets the number of highest-probability dictionary terms to store per cluster.

```
OutputFormat GetOutputFormat()
```

Returns a member of the OutputFormat enumerated type; this is the file format for the clustering results. The default output format is JSON.

```
void SetOutputFormat(const OutputFormat = OutputFormat::JSON)
```

Sets the output format for the clustering result file. The argument must be one of the values in the OutputFormat enumerated type.

```
double GetHierNmf2Tolerance()
```

Returns the tolerance value used by the NMF-RANK2 algorithm for hierarchical clustering. The default value is 1.0e-4.

```
void SetHierNmf2Tolerance(const double tol=1.0e-4)
```

Sets the tolerance value used by the NMF-RANK2 algorithm for hierarchical clustering. The tolerance value should satisfy 0.0 < tolerance < 1.0. Any inputs outside this range will cause a `std::logic_error` exception to be thrown.

```
void HierNmf2(const unsigned int num_clusters)
```

This function performs hierarchical clustering on the loaded matrix, generating the number of clusters specified by the `num_clusters` argument. For an overview of the hierarchical clustering process, see the description below for the hierclust command line application.

This function generates two output files in the output directory: *assignments_N.csv* and *tree_N.{json, xml}*. Here N is the number of clusters specified as an argument, and the tree file can be in either JSON XML format.

The content of the files is described below in the section on the hierclust command line application.

```
void HierNmf2WithFlat(const unsigned int num_clusters)
```

This function performs hierarchical clustering on the loaded matrix, exactly as described for HierNmf2. In addition, it also computes a flat clustering result. Thus four output files are generated. The flat clustering result files are `assignments_flat_N.csv` and `clusters_N.{json, xml}`. The cluster file contents are documented below in the section on the flatclust command line application.

CHAPTER

EIGHT

PYSMALLK API (PYTHON)

8.1 Introduction

Why Python? Although it's perfectly fine to run SmallK from the command line, Python provides a great deal more flexibility that augments the C++ code with other tasks that are much more easily accomplished with a very high level language. Python distributions can be easily extended with open source libraries from third party sources as well, two examples being numpy and scipy, well-known standards for scientific computing in the Python community. There are numerous packages available that extend these scientific libraries into the data analytics domain as well, such as scikit-learn[49].

For using scientific Python, we strongly recommend the Anaconda Python distribution provided by Continuum Analytics[50]. Download and installation instructions for all platforms can be found here[51]. Anaconda includes many if not most of the commonly used scientific and data analytics packages available and a very easy to use package manager and updating system. After installing Anaconda there will be available at the command line both a standard Python interpreter (type `python`) and an iPython interpreter (type `ipython`). We recommend using the iPython interpreter. In addition to the command line interfaces to Python, Anaconda includes the Spyder visual development environment featuring a very well thought out interface that makes developing Python code almost "too easy". Spyder has many features found in the Matlab™ editor and a similar look and feel.

Anaconda also includes the Cython package, which is used by SmallK to integrate the Python and C++ code. Cython[52] includes support for most of the C++ standard and supports the latest GNU C++ compilers. Most if not all the standard libraries are supported and the latest version (20.2) has support for the standard template library (STL) as well.

8.2 Examples of Pysmallk Usage

Pysmallk has five classes, each of which represents one of the SmallK tools: SmallkAPI (the simplistic Smallk API), Flatclust, Hierclust, Matrixgen, and Preprocessor. These tools can be strung together into various kind of applications. Examples of such applications can be found in `examples/pysmallk_example.py` and in the `pysmallk/tests/` subdirectory.

The smallk_data repository contains several files (`articles_matrix.mtx`, `articles_documents.txt`, `articles_dictionary.txt`) that contain the matrix and associated text files created from 2,424 news articles.

First, we will need to import numpy and the shared libary:

```
import numpy as np
import pysmallk
```

[49] http://scikit-learn.org/stable/index.html
[50] http://continuum.io/
[51] https://store.continuum.io/cshop/anaconda/
[52] http://cython.org/

We then should apply the preprocessor to our data:

```
p = pysmallk.Preprocessor()
p.load_matrix(filepath='smallk_data/articles_matrix.mtx')
p.load_dictionary(filepath='smallk_data/articles_dictionary.txt')
p.load_documents(filepath='smallk_data/articles_documents.txt')
```

We will begin with the default inputs and run preprocess:

```
p.preprocess()
```

Instead of writing the results to files, we can get the outputs from the Preprocessor class and pass them directly as inputs to the SmallkAPI class.:

```
reduced_docs = p.get_reduced_documents()
reduced_dict = p.get_reduced_dictionary()
reduced_scores = p.get_reduced_scores()
reduced_row_indices = p.get_reduced_row_indices()
reduced_col_offsets = p.get_reduced_col_offsets()
reduced_height = len(reduced_dict)
reduced_width = len(reduced_docs)
```

Now let's instantiate the SmallkAPI object that we will use to do further computations.:

```
sk = pysmallk.SmallkAPI()
```

One of the options for matrix loading is to pass in the appropriate fields for a sparse matrix, as so:

```
sk.load_matrix(buffer=reduced_scores, row_indices=reduced_row_indices, col_
↪offsets=reduced_col_offsets, height=reduced_height, width=reduced_width,␣
↪nz=len(reduced_scores))
```

The input matrix alone is sufficient to run NMF and compute the factor matricies.

```
sk.nmf(5, 'BPP')
```

This will compute the W and H factor matrices and subsequently write them to the files w.csv and h.csv, respectively.

We can continue with further calcuations using the same input matrix. For example, we can extract topic models from the input matrix if we also provide a dictionary.

```
sk.load_dictionary(dictionary=reduced_dict)
sk.hiernmf2(5)
```

This will use Hierarchical NMF to determine the final leaf nodes to use for the topic models and will output assignments_5.csv (cluster labels) and tree_5.xml.

Now let's say we want to create our own random matrix and pass that as a numpy matrix into SmallK.

```
a = np.random.random((256, 256))
```

In order to run the Hierclust or Flatclust applications, we will need to provide a dictionary file from which to select the top terms.

```
pathtodict = args.indir + 'reuters_dictionary.txt'
with open(pathtodict) as dictionary:
terms = dictionary.read().split("\n")
```

For illustration, let's use the Flatclust object and extract the resulting assignments from running NMF.

```
f = pysmallk.Flatclust()

f.load_matrix(matrix=a)
f.load_dictionary(dictionary=terms)
f.cluster(16, algorithm='HALS')
a = f.get_assignments()
```

Now the variable 'a' holds a list of the computed assignment labels for each of the 256 elements in our original matrix.

When we are finished, we should clean up the environment before exiting:

```
sk.finalize()
f.finalize()
```

8.3 Pysmallk Functions

Pysmallk has five classes, each of which represents one of the SmallK tools: SmallkAPI (the simplistic Smallk API), Flatclust, Hierclust, Matrixgen, and Preprocessor. Each of these classes can be imported as follows:

```
from pysmallk import SmallkAPI
from pysmallk import Flatclust
from pysmallk import Hierclust
from pysmallk import Matrixgen
from pysmallk import Preprocessor
```

Each class's primary functions are documented in the sections below. The parameters are either marked [in] or [kwarg] which represent, respectively, positional and keyword arguments.

8.3.1 Preprocessor

```
def parser()
```

Returns the parsed arguments for the default command line application. The command line arguments are the same as those for the C++ binary application preprocessor.

```
def load_matrix(filepath="", height=0, width=0, nz=0, buffer=[], row_indices=[], col_
↪offsets=[])
```

Load an input matrix.

1. To load a matrix from a file:

```
* filepath:      The path to the input matrix
```

2. To load a sparse matrix from Matrixgen:

```
* height:        The height of the sparse matrix
* width:         The width of the sparse matrix
* sparse_matrix: The sparse matrix returned from Matrixgen
```

3. To load a sparse matrix from python:

```
* height:        The height of the sparse matrix
* width:         The width of the sparse matrix
* nz:            The number of non-zeros in the sparse matrix
* buffer:        List of doubles containing the non-zero elements of the sparse matrix
* row_indices:   List of integers representing the row indices of the sparse matrix
* col_offsets:   List of integers representing the column offsets of the sparse matrix
```

```
def load_dictionary(filepath=None, dictionary=None)
```

Loads a dictionary from either a filepath or a list of dictionary strings.

```
def load_documents(filepath=None, documents=None)
```

Loads a documents from either a filepath or a list of document strings.

```
def get_reduced_documents()
```

Returns the reduced documents.

```
def get_reduced_dictionary()
```

Returns the reduced dictionary.

```
def get_reduced_scores()
```

Returns the non-zero scores from the reduced matrix.

```
def get_reduced_row_indices ()
```

Returns the row indices for the reduced matrix.

```
def get_reduced_col_offsets ()
```

Returns the column offsets for the reduced matrix.

```
def get_reduced_field (filepath="", values=[])
```

Loads a field from either a filepath or a list of field strings. Returns the reduced fields.

```
def preprocess(maxiter=1000, docsperterm=3,termsperdoc=5, boolean_mode=0)
```

Preprocesses the matrix.

- maxiter: The maximum number of iterations (optional)
- docsperterm: The number of documents required per term (optional)
- termsperdoc: The number of terms requried per document (optional)
- boolean_mode: All nonzero matrix elements will be treated as if they had the value 1.0 (optional)

```
def write_output(matrix_filepath, dict_filepath, docs_filepath, precision=4)
```

Writes the preprocessor results to files.

- matrix_filepath: The filepath for writing the matrix
- dict_filepath: The filepath for writing the dictionary
- docs_filepath: The filepath for the documents

- precision: The precision with which to write the outputs (optional)

8.3.2 Matrixgen

```
def parser()
```

Returns the parsed arguments for the default command line application. The command line arguments are the same as those for the C++ binary application matrixgen.

```
def uniform(m, n, center=0.5, radius=0.5)
```

Generates a uniform matrix. Returns a tuple of the list of values, the height, and the width.

- m: The desired height
- n: The desired width
- center: Center with which to initialize the RNG
- radius: Radius with which to initialize the RNG

```
def densediag(m, n, center=0.5, radius=0.5)
```

Generates a dense diagonal matrix. Returns a tuple of the list of values, the height, and the width.

- m: The desired height
- n: The desired width
- center: Center with which to initialize the RNG
- radius: Radius with which to initialize the RNG

```
def identify(m, n)
```

Generates an identify matrix. Returns a tuple of the list of values, the height, and the width.

- m: The desired height
- n: The desired width

```
def sparsediag(n, center=0.5, radius=0.5)
```

Generates a sparse diagonal matrix. Returns a tuple of the list of values, the height, and the width.

- n: The desired width
- center: Center with which to initialize the RNG
- radius: Radius with which to initialize the RNG

```
def ones(m, n)
```

Generates a matrix of ones. Returns a tuple of the list of values, the height, and the width.

- m: The desired height
- n: The desired width

```
def zeros(m, n)
```

Generates a matrix of zeros. Returns a tuple of the list of values, the height, and the width.

- m: The desired height
- n: The desired width

```
def sparse(m, n, nz)
```

Generates a random sparse matrix. Returns a tuple of the list of values, the height, and the width.

- m: The desired height
- n: The desired width
- nz: The number of non zeros in the matrix

```
def write_output(filename, precision=6)
```

Writes the generated matrix to file.

- filename: The filepath for writing the matrix
- precision: The precision with which to write the matrix

8.3.3 SmallkAPI

```
def parser()
```

Returns the parsed arguments for the default command line application. The dictionary containing the parsed arguments.

```
def get_major_version()
```

Returns the major version of SmallK.

```
def get_minor_version()
```

Returns the minor version of SmallK.

```
def get_patch_level()
```

Returns the patch level of SmallK.

```
def get_version_string()
```

Returns a string representation of the version of SmallK.

```
def load_matrix(filepath="", height=0, width=0, delim="", buffer=[], matrix=[],
nz=0, row_indices=[], col_offsets=[], column_major=False, sparse_matrix=None):
```

Load an input matrix.

1. To load a matrix from a file:
   - filepath: The path to the input matrix
2. To load a sparse matrix from python:
   - height: The height of the sparse matrix
   - width: The width of the sparse matrix
   - nz: The number of non-zeros in the sparse matrix

- buffer: List of doubles containing the non-zero elements of the sparse matrix
- row_indices: List of integers representing the row indices of the sparse matrix
- col_offsets: List of integers representing the column offsets of the sparse matrix

3. To load a dense matrix from python:
   - height: The height of the dense matrix
   - width: The width of the dense matrix
   - buffer: List of doubles containing the elements of the dense matrix
4. To load a numpy matrix from python:
   - matrix: The numpy matrix
   - column_major: Boolean for whether or not the matrix is column major (optional)

Note: Internal to SmallK, the matrix is stored in column-major order. When you are loading a numpy matrix, the assumption is that your matrix is in row-major order. If this is not the case, you can pass `column_major=True` in as a keyword argument. When directly loading a dense matrix, the assumption is that your buffer holds the data in column-major order as well.

```
def is_matrix_loaded()
```

Indicates whether or not a matrix has been loaded.

```
def nmf(k, algorithm, infile_W="", infile_H="", precision=4, min_iter=5, max_
↪iter=5000, tol=0.005, max_threads=8, outdir=".")
```

Runs NMF on the loaded matrix using the supplied algorithm and implementation details.

- k: The desired number of clusters
- algorithm: The desired NMF algorithm
- initdir: Initialization for W and H for each leaf (optional)
- precision: Precision for calcuations (optional)
- min_iter: Minimum number of iterations (optional)
- max_iter: Maximum number of iterations (optional)
- tol: Tolerance for determing convergence (optional)
- max_threads: Maximum number of threads to use (optional)
- outdir: Output directory for files (optional)

```
def get_inputs()
```

Returns a dictionary of the supplied inputs to the nmf function.

```
def get_H()
```

Returns the output H matrix.

```
def get_W()
```

Returns the output W matrix.

```
def load_dictionary (filepath="", dictionary=[])
```

Loads a dictionary from either a filepath or a list of dictionary strings.

```
def hiernmf2(k, format="XML", maxterms=5, tol=0.0001)
```

Runs HierNMF2 on the loaded matrix.

- k: The desired number of clusters
- format: Output format, XML or JSON (optional)
- maxterms: Maximum number of terms (optional)
- tol: Tolerance to use for determining convergence (optional)

```
def finalize()
```

Cleans up the elemental and smallk environment.

8.3.4 Flatclust

```
def parser()
```

Returns the parsed arguments for the default command line application. The command line arguemnts are the same as those for the C++ binary application flatclust.

```
def load_matrix(**kwargs)
```

Load an input matrix.

1. To load a matrix from a file:
   - filepath: The path to the input matrix
2. To load a sparse matrix from python:
   - height: The height of the sparse matrix
   - width: The width of the sparse matrix
   - nz: The number of non-zeros in the sparse matrix
   - buffer: List of doubles containing the non-zero elements of the sparse matrix
   - row_indices: List of integers representing the row indices of the sparse matrix
   - col_offsets: List of integers representing the column offsets of the sparse matrix
3. To load a sparse matrix from Matrixgen:
   - height: The height of the sparse matrix
   - width: The width of the sparse matrix
   - sparse_matrix: The sparse matrix returned from Matrixgen
4. To load a dense matrix from python:
   - height: The height of the dense matrix
   - width: The width of the dense matrix
   - buffer: List of doubles containing the elements of the dense matrix

5. To load a numpy matrix from python:
    - matrix: The numpy matrix
    - column_major: Boolean for whether or not the matrix is column major (optional)

Note: Internal to SmallK, the matrix is stored in column-major order. When you are loading a numpy matrix, the assumption is that your matrix is in row-major order. If this is not the case, you can pass `column_major=True` in as a keyword argument. When directly loading a dense matrix, the assumption is that your buffer holds the data in column-major order as well.

```
def load_dictionary (filepath="", dictionary=[])
```

Loads a dictionary from either a filepath or a list of dictionary strings.

```
def cluster(k, infile_W='', infile_H='', algorithm="BPP", maxterms=5, verbose=True,␣
↪min_iter=5, max_iter=5000, max_threads=8, tol=0.0001)
```

Runs NMF on the loaded matrix using the supplied algorithm and implementation details.

- k: The desired number of clusters
- infile_W: Initialization for W (optional)
- infile_H: Initialization for H (optional)
- algorithm: The desired NMF algorithm (optional)
- maxterms: Maximum number of terms per cluster (optional)
- verbose: Boolean for whether or not to be verbose (optional)
- min_iter: Minimum number of iterations (optional)
- max_iter: Maximum number of iterations (optional)
- max_threads: Maximum number of threads to use (optional)
- tol: Tolerance for determing convergence (optional)

```
def get_top_indices()
```

Return the top term indices for each cluster. The length of the returned array is maxterms*k, with the first maxterms elements belonging to the first cluster, the second maxterms elements belonging to the second cluster, etc.

```
def get_top_terms()
```

Return the top terms for each cluster.The length of the returned array is maxterms*k, with the first maxterms elements belonging to the first cluster, the second maxterms elements belonging to the second cluster, etc.

```
def get_assignments()
```

Return the list of cluster assignments for each document.

```
def write_output(assignfile, treefile, outdir='./', format='XML')
```

Writes the flatclust results to files.

- assignfile: The filepath for writing assignments
- fuzzyfile: The filepath for writing fuzzy assignments

- treefile: The filepath for the tree results
- outdir: The output directory for the output files (optional)
- format: The output format JSON or XML (optional)

```
def finalize()
```

Cleans up the elemental and smallk environment.

8.3.5 Hierclust

```
def parser()
```

Returns the parsed arguments for the default command line application. The command line arguemnts are the same as those for the C++ binary application hierclust.

```
def load_matrix(**kwargs)
```

Load an input matrix.

1. To load a matrix from a file:
   - filepath: The path to the input matrix
2. To load a sparse matrix from python:
   - height: The height of the sparse matrix
   - width: The width of the sparse matrix
   - nz: The number of non-zeros in the sparse matrix
   - buffer: List of doubles containing the non-zero elements of the sparse matrix
   - row_indices: List of integers representing the row indices of the sparse matrix
   - col_offsets: List of integers representing the column offsets of the sparse matrix
3. To load a sparse matrix from Matrixgen:
   - height: The height of the sparse matrix
   - width: The width of the sparse matrix
   - sparse_matrix: The sparse matrix returned from Matrixgen
4. To load a dense matrix from python:
   - height: The height of the dense matrix
   - width: The width of the dense matrix
   - buffer: List of doubles containing the elements of the dense matrix
5. To load a numpy matrix from python:
   - matrix: The numpy matrix
   - column_major: Boolean for whether or not the matrix is column major (optional)

Note: Internal to SmallK, the matrix is stored in column-major order. When you are loading a numpy matrix, the assumption is that your matrix is in row-major order. If this is not the case, you can pass `column_major=True`

in as a keyword argument. When directly loading a dense matrix, the assumption is that your buffer holds the data in column-major order as well.

```
def load_dictionary (filepath="", dictionary=[])
```

Loads a dictionary from either a filepath or a list of dictionary strings.

```
def cluster(k, initdir='', maxterms=5, unbalanced=0.1, trial_allowance=3,
↪verbose=True, flat=0, min_iter=5, max_iter=5000, max_threads=8, tol=0.0001)
```

Runs NMF on the loaded matrix using the supplied algorithm and implementation details.

- k: The desired number of clusters
- initdir: Initialization for W,H for each k (optional)
- maxterms: Maximum number of terms per cluster (optional)
- unbalanced: Unbalanced parameter (optional)
- trial_allowance: Number of trials to use (optional)
- verbose: Boolean for whether or not to be verbose (optional)
- flat: Whether or not to flatten the results (optional)
- min_iter: Minimum number of iterations (optional)
- max_iter: Maximum number of iterations (optional)
- max_threads: Maximum number of threads to use (optional)
- tol: Tolerance for determing convergence (optional)

```
def get_top_indices()
```

Return the top term indices for each cluster. The length of the returned array is maxterms*k, with the first maxterms elements belonging to the first cluster, the second maxterms elements belonging to the second cluster, etc.

```
def get_assignments()
```

Return the list of cluster assignments for each document.

```
def write_output(assignfile, fuzzyfile, treefile, outdir='./', format='XML')
```

Writes the flatclust results to files.

- assignfile: The filepath for writing assignments
- fuzzyfile: The filepath for writing fuzzy assignments
- treefile: The filepath for the tree results
- outdir: The output directory for the output files (optional)
- format: The output format JSON or XML (optional)

```
def finalize()
```

Cleans up the elemental and smallk environment.

CHAPTER

NINE

TESTS

Below we provide example output for many of the tests that can be run in the SmallK library. This will provide guidance for identifying issues with your installation. Of course results will vary on different machines.

9.1 SmallK Test Results

After building the smallk library, the *make check* command will run a bash script that performs a series of tests on the code. This is a sample output of those tests:

```
Build configuration: release
sh tests/scripts/test_smallk.sh ../xdata_github/smallk_data/ | tee smallk_test_
↪results.txt
*****************************************************
*                                                   *
*           Testing the smallk interface.           *
*                                                   *
*****************************************************
WARNING: Could not achieve THREAD_MULTIPLE support.
Smallk major version: 1
Smallk minor version: 6
Smallk patch level:   0
Smallk version string: 1.6.0
Loading matrix...

Running NMF-BPP...

Initializing matrix W...
Initializing matrix H...

                parameters:

                 algorithm: Nonnegative Least Squares with Block Principal Pivoting
        stopping criterion: Ratio of Projected Gradients
                    height: 12411
                     width: 7984
                         k: 8
                  miniter: 1
                  maxiter: 5000
                      tol: 0.005
                matrixfile: ../xdata_github/smallk_data/reuters.mtx
                maxthreads: 8

1:      progress metric:        (min_iter)
```

```
2:      progress metric:        0.35826
3:      progress metric:        0.172127
4:      progress metric:        0.106297
5:      progress metric:        0.0696424
6:      progress metric:        0.0538889
7:      progress metric:        0.0559478
8:      progress metric:        0.0686117
9:      progress metric:        0.0788641
10:     progress metric:        0.0711522
20:     progress metric:        0.00568349

Solution converged after 22 iterations.

Elapsed wall clock time: 0.633 sec.

Writing output files...

Running HierNmf2...

Loading dictionary...

                parameters:

                    height: 12411
                     width: 7984
                matrixfile: ../xdata_github/smallk_data/reuters.mtx
                  dictfile: ../xdata_github/smallk_data/reuters_dictionary.txt
                       tol: 0.0001
                   miniter: 1
                   maxiter: 5000
                  maxterms: 5
                maxthreads: 8
[1] [2] [3] [4]

Elapsed wall clock time: 551 ms.
9/9 factorizations converged.

Writing output files...
W matrix test passed
H matrix test passed
*****************************************************
*                                                   *
*            Testing the preprocessor.              *
*                                                   *
*****************************************************

      Command line options:

                     indir: ../xdata_github/smallk_data/
                    outdir: current directory
             docs_per_term: 3
             terms_per_doc: 5
                  max_iter: 1000
                 precision: 4
              boolean_mode: 0

Loading input matrix ../xdata_github/smallk_data/matrix.mtx
        Input file load time: 1.421s.
```

```
Starting iterations...
        [1] height: 39771, width: 11237, nonzeros: 877453
Iterations finished.
        New height: 39727
        New width: 11237
        New nonzero count: 877374
Processing time: 0.063s.

Writing output matrix 'reduced_matrix.mtx'
Output file write time: 2.189s.
Writing dictionary file 'reduced_dictionary.txt'
Writing documents file 'reduced_documents.txt'
Dictionary + documents write time: 0.083s.
preprocessor matrix test passed
preprocessor dictionary test passed
preprocessor documents test passed
**************************************************
*                                                *
*            Testing the NMF routines.           *
*                                                *
**************************************************
WARNING: Could not achieve THREAD_MULTIPLE support.
Loading matrix...
Initializing matrix W...
Initializing matrix H...

      Command line options:

                algorithm: Nonnegative Least Squares with Block Principal Pivoting
        stopping criterion: Ratio of Projected Gradients
                   height: 12411
                    width: 7984
                        k: 8
                  miniter: 1
                  maxiter: 5000
                      tol: 0.005
                 tolcount: 1
                  verbose: 1
                normalize: 1
             outprecision: 6
               matrixfile: ../xdata_github/smallk_data//reuters.mtx
                 infile_W: ../xdata_github/smallk_data//nmf_init_w.csv
                 infile_H: ../xdata_github/smallk_data//nmf_init_h.csv
                outfile_W: w.csv
                outfile_H: h.csv
               maxthreads: 8

1:      progress metric:        (min_iter)
2:      progress metric:        0.35826
3:      progress metric:        0.172127
4:      progress metric:        0.106297
5:      progress metric:        0.0696424
6:      progress metric:        0.0538889
7:      progress metric:        0.0559478
8:      progress metric:        0.0686117
9:      progress metric:        0.0788641
10:     progress metric:        0.0711522
```

```
20:     progress metric:        0.00568349

Solution converged after 22 iterations.

Elapsed wall clock time: 0.673 sec.

Writing output files...
NMF W matrix test passed
NMF H matrix test passed
*****************************************************
*                                                   *
*                Testing hierclust.                 *
*                                                   *
*****************************************************
-------------------------------

  Reuters matrix, 12 clusters

-------------------------------
WARNING: Could not achieve THREAD_MULTIPLE support.
loading dictionary...
loading matrix...

     Command line options:

                   height: 12411
                    width: 7984
               matrixfile: ../xdata_github/smallk_data//reuters.mtx
                  initdir: ../xdata_github/smallk_data//test/matrices.reuters/
                 dictfile: ../xdata_github/smallk_data//reuters_dictionary.txt
               assignfile: assignments_12.csv
                   format: XML
                 treefile: tree_12.xml
                 clusters: 12
                      tol: 0.0001
                   outdir:
                  miniter: 1
                  maxiter: 5000
                 maxterms: 5
               maxthreads: 8
               unbalanced: 0.1
          trial_allowance: 3
                     flat: 0
                  verbose: 1

[1] [2] [3] [4] [5] [6] dropping 20 items ...
[7] [8] [9] [10] [11]

Elapsed wall clock time: 2.758 s.
26/26 factorizations converged.

Writing output files...
XML file test passed
assignment file test passed
-------------------------------

  20News matrix, 15 clusters
```

```
------------------------------
WARNING: Could not achieve THREAD_MULTIPLE support.
loading dictionary...
loading matrix...

     Command line options:

                    height: 39727
                     width: 11237
                matrixfile: ../xdata_github/smallk_data//news20.mtx
                   initdir: ../xdata_github/smallk_data//test/matrices.20news/
                  dictfile: ../xdata_github/smallk_data//news20_dictionary.txt
                assignfile: assignments_15.csv
                    format: XML
                  treefile: tree_15.xml
                  clusters: 15
                       tol: 0.0001
                    outdir:
                   miniter: 1
                   maxiter: 5000
                  maxterms: 5
                maxthreads: 8
                unbalanced: 0.1
           trial_allowance: 3
                      flat: 0
                   verbose: 1

[1] [2] [3] dropping 30 items ...
[4] [5] dropping 132 items ...
[6] [7] [8] [9] [10] dropping 41 items ...
[11] dropping 51 items ...
[12] dropping 22 items ...
[13] dropping 85 items ...
[14]

Elapsed wall clock time: 10.308 s.
41/41 factorizations converged.

Writing output files...
XML file test passed
assignment file test passed
**************************************************
*                                                *
*              Testing flatclust.                *
*                                                *
**************************************************
WARNING: Could not achieve THREAD_MULTIPLE support.
loading dictionary...
loading matrix...
Initializing matrix W...
Initializing matrix H...

     Command line options:

                    height: 256
                     width: 256
                matrixfile: ../xdata_github/smallk_data//rnd_256_256.csv
                  infile_W: ../xdata_github/smallk_data//flatclust_init_w.csv
```

```
                  infile_H: ../xdata_github/smallk_data//flatclust_init_h.csv
                  dictfile: ../xdata_github/smallk_data//reuters_dictionary.txt
                assignfile: assignments_16.csv
                 fuzzyfile: assignments_fuzzy_16.csv
                    format: XML
                 clustfile: clusters_16.xml
                 algorithm: HALS
                  clusters: 16
                       tol: 0.0001
                    outdir:
                   miniter: 1
                   maxiter: 5000
                  maxterms: 5
                maxthreads: 8
                   verbose: 1

1:      progress metric:        (min_iter)
2:      progress metric:        0.635556
3:      progress metric:        0.490817
4:      progress metric:        0.479135
5:      progress metric:        0.474986
6:      progress metric:        0.44968
7:      progress metric:        0.422542
8:      progress metric:        0.407662
9:      progress metric:        0.395145
10:     progress metric:        0.379238
20:     progress metric:        0.272868
30:     progress metric:        0.168386
40:     progress metric:        0.109147
50:     progress metric:        0.0767327
60:     progress metric:        0.0488545
70:     progress metric:        0.036226
80:     progress metric:        0.0307648
90:     progress metric:        0.0266116
100:    progress metric:        0.0226963
110:    progress metric:        0.0188616
120:    progress metric:        0.0158307
130:    progress metric:        0.0137605
140:    progress metric:        0.0127888
150:    progress metric:        0.0123962
160:    progress metric:        0.0124734
170:    progress metric:        0.0123563
180:    progress metric:        0.0122163
190:    progress metric:        0.0120643
200:    progress metric:        0.0117647
210:    progress metric:        0.0114894
220:    progress metric:        0.0110467
230:    progress metric:        0.0107816
240:    progress metric:        0.0105239
250:    progress metric:        0.0103824
260:    progress metric:        0.0100915
270:    progress metric:        0.00965073
280:    progress metric:        0.00938526
290:    progress metric:        0.00914129
300:    progress metric:        0.00896701
310:    progress metric:        0.00886729
320:    progress metric:        0.00841059
330:    progress metric:        0.007793
```

```
340:    progress metric:        0.00740095
350:    progress metric:        0.00708869
360:    progress metric:        0.00683069
370:    progress metric:        0.00672093
380:    progress metric:        0.00687906
390:    progress metric:        0.00703777
400:    progress metric:        0.00721928
410:    progress metric:        0.00729384
420:    progress metric:        0.00718332
430:    progress metric:        0.00722893
440:    progress metric:        0.00726766
450:    progress metric:        0.00739665
460:    progress metric:        0.00769819
470:    progress metric:        0.00814673
480:    progress metric:        0.008566
490:    progress metric:        0.00877955
500:    progress metric:        0.00884221
510:    progress metric:        0.0088057
520:    progress metric:        0.00852345
530:    progress metric:        0.00797952
540:    progress metric:        0.00749354
550:    progress metric:        0.00689316
560:    progress metric:        0.00623287
570:    progress metric:        0.00576619
580:    progress metric:        0.00541125
590:    progress metric:        0.00501715
600:    progress metric:        0.00466547
610:    progress metric:        0.00432811
620:    progress metric:        0.00412669
630:    progress metric:        0.00383406
640:    progress metric:        0.00352802
650:    progress metric:        0.00331556
660:    progress metric:        0.00315735
670:    progress metric:        0.00304253
680:    progress metric:        0.00296627
690:    progress metric:        0.00289013
700:    progress metric:        0.00279647
710:    progress metric:        0.00271036
720:    progress metric:        0.00261087
730:    progress metric:        0.0025158
740:    progress metric:        0.00245123
750:    progress metric:        0.00237435
760:    progress metric:        0.00231126
770:    progress metric:        0.00228199
780:    progress metric:        0.00227623
790:    progress metric:        0.00228185
800:    progress metric:        0.00227993
810:    progress metric:        0.00228216
820:    progress metric:        0.00228018
830:    progress metric:        0.00229096
840:    progress metric:        0.00232403
850:    progress metric:        0.00234957
860:    progress metric:        0.00227868
870:    progress metric:        0.00210786
880:    progress metric:        0.00195462
890:    progress metric:        0.00183587
900:    progress metric:        0.00173358
910:    progress metric:        0.0016405
```

```
920:    progress metric:        0.00156422
930:    progress metric:        0.00150835
940:    progress metric:        0.00146594
950:    progress metric:        0.00143261
960:    progress metric:        0.00137378
970:    progress metric:        0.00131989
980:    progress metric:        0.00126626
990:    progress metric:        0.0012164
1000:   progress metric:        0.00117061
1010:   progress metric:        0.00112539
1020:   progress metric:        0.00108626
1030:   progress metric:        0.00105192
1040:   progress metric:        0.00102131
1050:   progress metric:        0.000992069
1060:   progress metric:        0.000965259
1070:   progress metric:        0.000938949
1080:   progress metric:        0.000911962
1090:   progress metric:        0.000884505
1100:   progress metric:        0.000854904
1110:   progress metric:        0.000820121
1120:   progress metric:        0.000785245
1130:   progress metric:        0.000752513
1140:   progress metric:        0.000723279
1150:   progress metric:        0.000697698
1160:   progress metric:        0.000680904
1170:   progress metric:        0.000652152
1180:   progress metric:        0.000628268
1190:   progress metric:        0.000612413
1200:   progress metric:        0.000596834
1210:   progress metric:        0.000580674
1220:   progress metric:        0.000556549
1230:   progress metric:        0.000535666
1240:   progress metric:        0.00051492
1250:   progress metric:        0.000496234
1260:   progress metric:        0.000481147
1270:   progress metric:        0.000461294
1280:   progress metric:        0.000440802
1290:   progress metric:        0.000419049
1300:   progress metric:        0.000398007
1310:   progress metric:        0.000376203
1320:   progress metric:        0.000355811
1330:   progress metric:        0.00033729
1340:   progress metric:        0.000318932
1350:   progress metric:        0.000302528
1360:   progress metric:        0.000287961
1370:   progress metric:        0.00027486
1380:   progress metric:        0.00026403
1390:   progress metric:        0.000255504
1400:   progress metric:        0.000248646
1410:   progress metric:        0.000242996
1420:   progress metric:        0.000239243
1430:   progress metric:        0.000236852
1440:   progress metric:        0.000235313
1450:   progress metric:        0.000234465
1460:   progress metric:        0.000234154
1470:   progress metric:        0.000234253
1480:   progress metric:        0.00023487
1490:   progress metric:        0.000237223
```

```
1500:   progress metric:        0.000240043
1510:   progress metric:        0.000243896
1520:   progress metric:        0.00024867
1530:   progress metric:        0.000253981
1540:   progress metric:        0.000260239
1550:   progress metric:        0.000266795
1560:   progress metric:        0.000273529
1570:   progress metric:        0.000280678
1580:   progress metric:        0.000287273
1590:   progress metric:        0.000292288
1600:   progress metric:        0.000296475
1610:   progress metric:        0.000299556
1620:   progress metric:        0.00030244
1630:   progress metric:        0.000306148
1640:   progress metric:        0.000310299
1650:   progress metric:        0.000314674
1660:   progress metric:        0.000319052
1670:   progress metric:        0.000323906
1680:   progress metric:        0.000329536
1690:   progress metric:        0.000335913
1700:   progress metric:        0.000342834
1710:   progress metric:        0.000351167
1720:   progress metric:        0.000352515
1730:   progress metric:        0.000348749
1740:   progress metric:        0.000345684
1750:   progress metric:        0.000343139
1760:   progress metric:        0.000340867
1770:   progress metric:        0.000339052
1780:   progress metric:        0.000337038
1790:   progress metric:        0.000335244
1800:   progress metric:        0.000333452
1810:   progress metric:        0.000332111
1820:   progress metric:        0.000330198
1830:   progress metric:        0.000325983
1840:   progress metric:        0.000321473
1850:   progress metric:        0.000316999
1860:   progress metric:        0.000312054
1870:   progress metric:        0.000305176
1880:   progress metric:        0.000294684
1890:   progress metric:        0.000284482
1900:   progress metric:        0.000274905
1910:   progress metric:        0.000265684
1920:   progress metric:        0.000256761
1930:   progress metric:        0.000248203
1940:   progress metric:        0.000239613
1950:   progress metric:        0.000230677
1960:   progress metric:        0.00022218
1970:   progress metric:        0.000214089
1980:   progress metric:        0.00020621
1990:   progress metric:        0.000196915
2000:   progress metric:        0.000187712
2010:   progress metric:        0.000179199
2020:   progress metric:        0.00017137
2030:   progress metric:        0.000164158
2040:   progress metric:        0.000157751
2050:   progress metric:        0.000152485
2060:   progress metric:        0.000147217
2070:   progress metric:        0.000142083
```

```
2080:   progress metric:        0.000137148
2090:   progress metric:        0.000132379
2100:   progress metric:        0.000127922
2110:   progress metric:        0.000123617
2120:   progress metric:        0.000119548
2130:   progress metric:        0.000115684
2140:   progress metric:        0.000111997
2150:   progress metric:        0.000108389
2160:   progress metric:        0.000104838
2170:   progress metric:        0.000101387

Solution converged after 2175 iterations.

Elapsed wall clock time: 1.022 sec.

XML file test passed
assignment file test passed
fuzzy assignment file test passed
***** SmallK: All tests passed. *****
```

CHAPTER

TEN

BENCHMARKS AND RESULTS

Page under construction

CHAPTER

ELEVEN

PUBLICATIONS

1. Jingu Kim and Haesun Park, “Fast nonnegative matrix factorization: An active-set-like method and comparisons”, SIAM Journal on Scientific Computing, 33(6), pp. 3261-3281, 2011.[53]
2. Jingu Kim, Yunlong He, and Haesun Park, “Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework”, Journal of Global Optimization, 2013.[54]
3. Jaegul Choo, Changhyun Lee, Chandan K. Reddy, and Haesun Park, “UTOPIAN: User-driven Topic modeling based on interactive nonnegative matrix factorization”, IEEE Transactions on Visualization and Computer Graphics (TVCG), 19-12, pages 1992-2001, 2013.[55]
4. Da Kuang and Haesun Park, “Fast Rank-2 Nonnegative Matrix Factorization for Hierarchical Document Clustering”, Proceedings 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ‘13), 2013.[56]
5. Nicolas Gillis, Da Kuang and Haesun Park, “Hierarchical clustering of hyperspectral images using rank-two nonnegative matrix factorization”, IEEE Transactions on Geoscience and Remote Sensing, 2014.[57]
6. Jaegul Choo, Barry Drake, and Haesun Park, “Visual analytics for interactive exploration of large-scale document data via Nonnegative Matrix Factorization”, Proceedings for BigData Innovators Gathering (BIG) 2014, co-located with WWW2014, Seoul, Korea, 2014.[58]
7. R. Kannan, M. Ishteva, B. Drake, and H. Park, Bounded Matrix Low Rank Approximation, chapter in Nonnegative Matrix Factorization Techniques: Advances in Theory and Applications, Ganesh R. Naik (Ed.), Signals and Communication Technology Series, Springer-Verlag Berlin Heidelberg, 2016. DOI: 10.1007/978-3-662-48331-2, ISBN 978-3-662-48331-2[59]
8. Minsuk Choi, Jaesong Yoo, Ashley S. Beavers, Scott Longevin, Chris Bethune, Sean McIntyre, Barry Drake, Jaegul Choo, Haesun Park. “Tile-based Spatio-Temporal Visual Analytics via Topic Modeling on Social Media”, IEEE Vis 2016, Baltimore, Maryland, USA Oct 23-28, 2016.[60]
9. Rundong Du, Da Kuang, Barry Drake, and Haesun Park. “DC-NMF: Nonnegative Matrix Factorization based on Divide-and-Conquer for Fast Clustering and Topic Modeling”, Journal of Global Optimization, April 2017.[61]
10. Barry Drake, Tiffany Huang, Ashley Scripka Beavers, Rundong Du, and Haesun Park. “Event Detection based on Nonnegative Matrix Factorization: Ceasefire Violation, Environmental, and Malware Events”, Proceedings of the 8th International Conference on Applied Human Factors and Ergonomics (AHFE 2017), The Westin

[53] https://smallk.github.io/papers/SISC_082117RR_Kim_Park.pdf
[54] https://smallk.github.io/papers/nmf_review_jgo.pdf
[55] https://smallk.github.io/papers/2013_tvcg_utopian.pdf
[56] https://smallk.github.io/papers/hierNMF2.pdf
[57] https://smallk.github.io/papers/HierNMFImage.pdf
[58] https://smallk.github.io/papers/big_vanmf.pdf
[59] https://link.springer.com/chapter/10.1007%2F978-3-662-48331-2_4
[60] https://uncharted.software/assets/tile-based-topic-modeling.pdf
[61] https://link.springer.com/article/10.1007/s10898-017-0515-z

Bonaventure Hotel, Los Angeles, California, USA, AHFE, Human Factors in Cyber Warfare, (invited paper), July 17-21, DOI 10.1007/978-3-319-60585-2_16, Springer, 2017.[62]

11. Rundong Du, Barry Drake, and Haesun Park. "Hybrid Clustering based on Content and Connection Structure using Joint Nonnegative Matrix Factorization". Journal of Global Optimization, October, 2017.[63]

12. Sungbok Shin, Minsuk Choi, Jinho Choi, Scott Langevin, Christopher Bethune, Philippe Horne, Nathan Kronenfeld, Ramakrishnan Kannan, Barry Drake, Haesun Park, and Jaegul Choo, "STExNMF: Spatio-Temporally Exclusive Topic Discovery for Anomalous Event Detection", submitted, IEEE International Conference on Data Mining, New Orleans, USA, November 18-21, 2017.

If the SmallK library is used to obtain results for publication, please use the following BibTeX citation:

```
@misc{SmallK,
        author       = {Barry Drake and Stephen Lee-Urban and Haesun Park},
        title        = {SmallK is a {C++}{/}{P}ython high-performance software
                                    library for nonnegative matrix factorization␣
↪(NMF)
                                    and hierarchical and flat clustering using
                                    the NMF; current version 1.6.2},
        howpublished = {\url{http://smallk.github.io/}},
        month        = {June},
        year         = {2017}
}
```

[62] https://link.springer.com/chapter/10.1007/978-3-319-60585-2_16
[63] https://link.springer.com/article/10.1007/s10898-017-0578-x

CHAPTER

TWELVE

SOFTWARE REPO

12.1 Getting the code and instructions

These are the ways to obtain the code:

1. The SmallK code tarball is at SmallK[64].
2. There is also a code repository here[65].
3. And a data repository smallk_data[66]. The data repository will be needed to run the tests and example code.

Installation and build instructions can be found at:

1. Online installation instructions[67].
2. A pdf of the installation instructions[68] is also included at these sites.

12.2 Contact Info

For comments, questions, bug reports, suggestions, etc., contact:

Barry Drake
Research Scientist
Information and Communications Laboratory (ICL)
Information and Cyber Sciences Directorate (ICSD)
Georgia Tech Research Institute (GTRI)
75 5TH St. NW STE 900
ATLANTA, GA 30308-1018
barry.drake@gtri.gatech.edu

Stephen Lee-Urban
Research Scientist
Information and Communications Laboratory (ICL)
Information and Cyber Sciences Directorate (ICSD)
Georgia Tech Research Institute (GTRI)

[64] https://github.com/smallk/smallk.github.io/tree/master/code
[65] https://github.com/smallk/smallk
[66] https://github.com/smallk/smallk_data
[67] http://smallk.github.io/index.html
[68] http://smallk.github.io/doc/smallk_readme.pdf

75 5TH St. NW STE 900
ATLANTA, GA 30308-1018
stephen.lee-urban@gtri.gatech.edu

CHAPTER

THIRTEEN

PARTNERS

[69]

[70]

[71]

[69] https://gtri.gatech.edu/
[70] http://www.gatech.edu/
[71] http://www.darpa.mil/