Final Project

Software Project (0368-2161) Due: 05.05.25

1 Introduction

In this project you will implement a clustering algorithm that is based on symmetric Non-negative Matrix Factorization (symNMF). You will further apply it to several datasets and compare to K-means. This document starts by introducing the mathematical basis and algorithms for the project, and then describes the code and implementation requirements.

SymNMF We present the SymNMF algorithm based on [1]. Given a set of n points $X = x_1, x_2, \ldots, x_N \in R^d$ the algorithm is:

Algorithm 1 SymNMF Algorithm

1: Form the similarity matrix A from X (see 1.1)
2: Compute the Diagonal Degree Matrix (see 1.2)
3: Compute the normalized similarity W (see 1.3)
4: Find $H_{n\times k}$ that solves: $\min_{H\geq 0}\|W - HH^T\|_F^2$ (see 1.4)

Where k is a parameter denoting the required number of clusters and $\|\|_F^2$ is the squared Frobenius norm.

1.1 The Similarity Matrix

The similarity matrix $A \in \mathbb{R}^{n\times n}$ is defined as:

$$a_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2}\right) \; if \; i \neq j, \; and \; a_{ii} = 0$$

We denote by $\|\|^2$ the squared Euclidean distance ($\|a-b\|^2 = \sum_{i=1}^{d}(a_i - b_i)^2$).

1.2 The diagonal degree Matrix

The degree matrix D is defined as the diagonal matrix with degrees $d_1, ..., d_n$ on the diagonal and zero elsewhere. The degree of a vertex $x_i \in X$ is defined as:

$$d_i = \sum_{j=1}^{n} a_{ij} \tag{1}$$

1.3 The normalized similarity matrix

The graph Laplacian $W \in \mathbb{R}^{n \times n}$ is defined as

$$W = D^{-1/2} A D^{-1/2}$$

1.4 Algorithm for optimizing H

In this section, we describe the steps of finding the decomposition matrix H. The Objective is to find a lower dimension non-negative matrix $H_{n \times k},\ k << n$, such:

$$\min_{H \geq 0} \|W - HH^T\|_F^2$$

1.4.1 Initialize H

Randomly initialize H with values from the interval $[0, 2 * sqrt(m/k)]$, where m is the average of all entries of W.

1.4.2 Update H

After initializing H which will be denoted as $H^{(0)}$ (superscripts denote iteration indices), we iteratively update $H^{(t)}$ using the following rule:

$$H_{ij}^{(t+1)} \leftarrow H_{ij}^{(t)} \left(1 - \beta + \beta \frac{(WH^{(t)})_{ij}}{(H^{(t)}(H^{(t)})^T H^{(t)})_{ij}}\right)$$

where $\beta = 0.5$

1.4.3 Convergence

Update H using the above rule until max iteration number is reached OR the
$\|H^{(t+1)} - H^{(t)}\|_F^2 < \epsilon$.

1.5 Deriving a clustering solution

H can be viewed as an association matrix that gives an association score to each element with every cluster. To derive a hard clustering, we choose for each element the cluster with the highest association score.

For example:

$$H = \begin{pmatrix} 0.0600 & 0.0100 \\ 0.0100 & 0.0500 \\ 0.0100 & 0.0400 \\ 0.0200 & 0.0400 \\ 0.0500 & 0.0200 \end{pmatrix}$$

We have 2 clusters (columns), row number 1 belongs to first cluster, because at first raw the maximum is 0.06 and it's at first column. Second row gets second cluster and so on for other rows.

2 Assignment Description

Implement the following files:

1. `symnmf.py`: Python interface of your code.
2. `symnmf.h`: C header file.
3. `symnmf.c`: C interface of your code.
4. `symnmfmodule.c`: Python C API wrapper.
5. `analysis.py`: Analyze the algorithm.
6. `setup.py`: The setup file.
7. `Makefile`: Your make script to build the C interface.
8. `*.c/h`: Other modules and headers per your design.

2.1 Python Program (`symnmf.py`)

1. Reading user CMD arguments:
   (a) k (int, $< N$): Number of required clusters.
   (b) `goal`: Can get the following values:
      i. `symnmf`: Perform full the symNMF as described in 1 and output H.
      ii. `sym`: Calculate and output the similarity matrix as described in 1.1.
      iii. `ddg`: Calculate and output the Diagonal Degree Matrix as described in 1.2.
      iv. `norm`: Calculate and output the normalized similarity matrix as described in 1.3.
   (c) `file_name` (.txt): The path to the Input file, it will contain **N data points for all above goals**, the file extension is .txt
2. Implementation of H initialization when the `goal=symnmf`, as detailed in 1.4.1:
   (a) Set `np.random.seed(1234)` **at the beginning of your code**.
   (b) Use `np.random.uniform()` for random selection.
3. Interfacing with your C extension:
   (a) Import C module `symnmf`
   (b) if the goal='symnmf', call the `symnmf()` method with passing the initial H, the W and other arguments if needed, and get the final H.
   (c) if the goal='sym', call the `sym()` method with passing the datapoints X, and get similarity matrix.
   (d) if the goal='ddg', call the `ddg()` method with passing the datapoints X, and get diagonal degree matrix.

(e) if the goal='norm', call the `norm()` method with passing the datapoints X, and get normalized similarity matrix.

4. Output the required matrix separated by a comma, such that each row is in a line of its own.

Example:

```
>>> python3 symnmf.py 2 symnmf input_1.txt
    0.0600,0.0100
    0.0100,0.0500
    0.0100,0.0400
    0.0200,0.0400
    0.0500,0.0200
```

2.2 C Program (`symnmf.c`)

This is the C implementation program, with the following requirements:

1. Reading user CMD arguments:

   (a) `goal`: Can get the following values:

      i. `sym`: Calculate and output the similarity matrix as described in 1.1.

      ii. `ddg`: Calculate and output the Diagonal Degree Matrix as described in 1.2.

      iii. `norm`: Calculate and output the normalized similarity matrix as described in 1.3.

   (b) `file_name` (.txt): The path to the Input file, it will contain **N data points for all above goals**, the file extension is .txt

2. Output the required matrix separated by a comma, such that each row is in a line of its own.

The program must compile cleanly (no errors, no warnings) when running the following command:

```
$make
```

After successful compilation the program can run for Example:

```
>>>./symnmf sym input_1.txt
0.0000,0.0447,0.0456,...,0.0706,0.3615,0.0425
0.0447,0.0000,0.2871,...,0.0004,0.1665,0.6122
0.0456,0.2871,0.0000,...,0.0013,0.0228,0.1858
...
0.0706,0.0004,0.0013,...,0.0000,0.0150,0.0036
0.3615,0.1665,0.0228,...,0.0150,0.0000,0.1899
0.0425,0.6122,0.1858,...,0.0036,0.1899,0.0000
```

2.3 Python C API (`symnmfmodule.c`)

Start the file with:

```c
#define PY_SSIZE_T_CLEAN
#include <Python.h>
```

In this file you will define your C extension which will serve the functions: `symnmf`,`sym`,`ddg`,`norm` for Python, see 3.

2.4 C Header file (`symnmf.h`)

This header have to define all functions prototypes that is being used in `symnmfmodule.c` and implemented at `symnmf.c`.

2.5 `analysis.py`

Compare SymNMF to Kmeans from HW1. Apply both methods to given dataset and report the silhouette_score from the sklearn.metrics. For SymNMF, cluster assignment is done as explained in 1.5.

The silhouette score measures clustering quality by comparing the between-cluster distance against within-cluster distance. A higher score indicates better-defined clusters. The score is calculated as the mean of the silhouette coefficient of each data point separately, which is computed by the formula below:

$$\text{Silhouette coefficient} = \frac{b - a}{\max(a, b)}$$

where:

- a represents the mean distance between a data point and all other points within its cluster C.
- b represents the minimum over all other clusters $D \neq C$ of the mean distance between the data point and all points in D.

The program has two arguments: file_name(.txt): The path to the Input file, it will contain N data points, the file extension is .txt, and the number of clusters k.

Example run with input file(input_k5_d7.txt):

```
>>> python3 analysis.py 5 input_k5_d7.txt
    nmf: 0.1162
    kmeans: 0.1147
```

You can assume that the file_name and k are legal values.

2.6 Setup (`setup.py`)

This is the build used to create the *.so file that will allow `symnmf.py` to import `symnmfmodule`.

2.7 `Makefile`

Make script for building `symnmf` executable, considering all it's dependency. The compilation command should include all the flags as below:

```
gcc -ansi -Wall -Wextra -Werror -pedantic-errors
```

2.8 Build and Running

1. The extension must build cleanly (no errors, no warnings) when running the following command:

   ```
   $python3 setup.py build_ext --inplace
   ```

2. After successful build, the program must run as detailed in example 2.1.

3. **Don't compile the C module with gcc.**

2.9 Assumptions

Note that the following list applies to all code in this assignment:

1. Your code must run on the course's Docker, implement it as described here.

2. No need to validate arguments.

3. Outputs must be formatted to 4 decimal places (use: '%.4f') in both languages, for example:

   - $8.88885 \Rightarrow 8.8888$
   - $5.92237098749999997906 \Rightarrow 5.9224$
   - $2.231 \Rightarrow 2.2310$

4. There is no test files for this projects, you can create ones and test yourself.

5. Handle errors as following:

   (a) In case of any error, print "An Error Has Occurred" and terminate.

6. Do not forget to free any memory you allocated.

7. You can assume that all given data points are different.

8. Use `double` in C and `float` in Python for all vector's elements.

9. For Kmeans and NMF convergence, use $\epsilon = 1e-4,\ max\_iter = 300$.

3 Submission

1. Please submit a file named `id1_id2_project.zip` via Moodle, where id1 and id2 are the ids of the partners.
   (a) In case of individual submission, id2 must be 111111111
2. Put the following files ONLY in a folder called id1_id2_project:
   (a) `symnmf.py`
   (b) `symnmf.c`
   (c) `symnmfmodule.c`
   (d) `symnmf.h`
   (e) `analysis.py`
   (f) `setup.py`
   (g) `Makefile`
   (h) more `*.c, *.h` (optional)
3. Zip the folder using the following Linux cmd:

```
$zip -r id1_id2_project.zip id1_id2_project
```

4. **There will be manual test of your code.** The code will be checked for modularity, documentation, function length (up to 40 lines, including documentation), etc.

References

[1] Da Kuang, Chris Ding, and Haesun Park. Symmetric nonnegative matrix factorization for graph clustering. In *Proceedings of the 2012 SIAM International Conference on Data Mining (SDM)*, Proceedings, pages 106–117. Society for Industrial and Applied Mathematics, April 2012.