Final Project

Software Project (0368-2161) Due: 16/04/2023 23:59

1 Introduction

In this project you will implement a version of the unnormalized spectral clustering algorithm. This document starts by introducing the mathematical basis and algorithms for the project, and then describes the code and implementation requirements.

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

Algorithm 1 The Unormalized Spectral Clustering Algorithm

```
1: Form the weighted adjacency matrix $W$ from $X$ (see details in 1.1)
2: Compute the graph Laplacian $L$ (see details in 1.1)
3: Compute the first $k$ eigenvectors $u_1, \dots, u_k$ of $L$ (see details in 1.2,1.3)
4: Let $U \in R^{n \times k}$ be the matrix containing the vectors $u_1, \dots, u_k$ of $L$ as columns.
5: Treating each row of $U$ as a point in $R^k$, cluster them into $k$ clusters using K-means++
```

Where:

- "the first k eigenvectors" we refer to the eigenvectors corresponding to the k smallest eigenvalues.

1.1 Graph Representation

We aim at creating an undirected graph $G = (V, E; W)$, that will represent the n points at hand. Each point x_i is viewed as a vertex v_i (also known as node) to produce $V = \{v_1, v_2, \dots, v_n\}$. A common mapping is to set $v_i = i$. The set of edges (also known as arcs) E will be the union of all connected pairs $\{v_i, v_j\}$. Next, we present the way to decide the weights of the graph and the structure of the graph.

1.1.1 The Weighted Adjacency Matrix

The weighted adjacency matrix $W \in R^{n \times n}$ (also referred to as the affinity matrix) is defined as:

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

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

1.1.2 The Diagonal Degree Matrix

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

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

1.1.3 The Graph Laplacian

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

$$L = D - W$$

1.2 Finding Eigenvalues and Eigenvectors

In this section, we lay the foundation needed to find **all** of the eigenvectors and eigenvalues of a real, symmetric, full rank matrix.

1.2.1 Jacobi algorithm

The Jacobi eigenvalue algorithm is an iterative method for the calculation of the eigenvalues and eigenvectors of a real symmetric matrix (a process known as diagonalization).

1. **Procedure**

   (a) Build a rotation matrix P (as explained below).

   (b) Transform the matrix A to:

   $$A' = P^T A P$$

   $$A = A'$$

   (c) Repeat a,b until A' is diagonal matrix.

   (d) The diagonal of the final A' is the eigenvalues of the original A.

   (e) Calculate the eigenvectors of A by multiplying all the rotation matrices:

   $$V = P_1 P_2 P_3 \dots$$

2. **Rotation Matrix** P

   Let A be a symmetric matrix and P be the Jacobi rotation matrix of the form:

$$P = \begin{pmatrix} 1 & & & & & & \\ & \dots & & & & & \\ & & c & \dots & s & & \\ & & \vdots & 1 & \vdots & & \\ & & -s & \dots & c & & \\ & & & & & \dots & \\ & & & & & & 1 \end{pmatrix}$$

Here all the diagonal elements are unity except for the two elements c in rows (and columns) i and j and all off-diagonal elements are zero except the two elements s and $-s$. Also, $s = \sin\phi$ and $c = \cos\phi$.

3. **Pivot**
   The A_{ij} is the off-diagonal element with **the largest absolute value**.

4. **Obtain** c, t

$$\theta = \cot 2\phi = \frac{A_{jj} - A_{ii}}{2A_{ij}}$$

$$t = \frac{sign(\theta)}{|\theta| + \sqrt{\theta^2 + 1}}$$

$$c = \frac{1}{\sqrt{t^2 + 1}}, \quad s = tc$$

   Note: We define $sign(0) = 1$

5. **Convergence**: Let $off(A)^2$ and $off(A')^2$ be the sum of squares of all off-diagonal elements of A and A' respectively. Then the square of off diagonal elements of A is

$$off(A)^2 = \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} a_{ij}^2$$

   At step c in the above procedure, we define convergence as follow:

$$off(A)^2 - off(A')^2 \leq \epsilon$$

   We will be using $\epsilon = 1.0 \times 10^{-5}$ OR maximum number of rotations = 100

6. **Relation between** A **and** A':
   After each transformation of step 2, the modified elements of A are only the i and j rows and columns. Therefore, from the symmetry of A, we obtain the following formula to calculate A':

$$\begin{aligned}
a'_{ri} &= ca_{ri} - sa_{rj} \qquad r \neq i, j \\
a'_{rj} &= ca_{rj} + sa_{ri} \qquad r \neq i, j \\
a'_{ii} &= c^2 a_{ii} + s^2 a_{jj} - 2sca_{ij} \\
a'_{jj} &= s^2 a_{ii} + c^2 a_{jj} + 2sca_{ij} \\
a'_{ij} &= (c^2 - s^2)a_{ij} + sc(a_{ii} - a_{jj}) \Rightarrow a'_{ij} = 0
\end{aligned}$$

   Note: A' **is always symmetric.**

1.3 The Eigengap Heuristic

In order to determine the number of clusters k, we will use eigengap heuristic as follow:
let $(\delta_i)_{i=1,...,n-1}$ be the eigengap for the ascending ordered eigenvalues $0 = \lambda_1 \leq \ldots \leq \lambda_n$ of L, defined as:

$$\delta_i = |\lambda_i - \lambda_{i+1}|$$

The eigengap measure could indicate the number of clusters k through the following estimation:

$$k = argmax_i(\delta_i), \quad i = 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor$$

In case of equality in the $argmax$ of some eigengaps, use the lowest index.

2 Assignment Description

Implement the following files:

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

2.1 Python Program (`spkmeans.py`)

1. Reading user CMD arguments:
   (a) k (int, $< N$)(Optional): Number of required clusters. Default use the heuristic 1.3.
   (b) `goal`: Can get the following values:
      i. `spk`: Perform full spectral kmeans as described in 1.
      ii. `wam`: Calculate and output the Weighted Adjacency Matrix as described in 1.1.1.
      iii. `ddg`: Calculate and output the Diagonal Degree Matrix as described in 1.1.2.
      iv. `gl`: Calculate and output the Graph Laplacian as described in 1.1.3.
      v. `jacobi`: Calculate and output the eigenvalues and eigenvectors as described in 1.2.1.
   (c) `file_name` (.txt): The path to the Input file, it will contain **N data points for all above goals except Jacobi, in case the goal is Jacobi the input is a symmetric matrix**, the file extension is .txt

2. Implementation of the k-means++ initialization when the goal=spk, as detailed in HW2:

    (a) Set np.random.seed(0) **at the beginning of your code**.

    (b) Use np.random.choice() for random selection.

3. Interfacing with your C extension:

    (a) Import C module mykmeanssp

    (b) if the goal='spk', call the spk() method with passing the initial centroids, the datapoints and other arguments if needed, and get the final centroids.

    (c) if the goal='wam', call the wam() method with passing the datapoints, and get weighted adj matrix.

    (d) if the goal='ddg', call the ddg() method with passing the datapoints, and get diagonal degree matrix.

    (e) if the goal='gl', call the gl() method with passing the datapoints, and get graph Laplacian matrix.

    (f) if the goal='jacobi', call the jacobi() method with passing the matrix, and get the eigenvalues and eigenvectors.

4. Outputting the following:

    - Case of 'spk': The first line will be the indices of the observations chosen by the K-means++ algorithm as the initial centroids. We refer to the first observation index as 0, the second as 1 and so on, up until N - 1. The second line onward will be the calculated final centroids from the K-means algorithm, separated by a comma, such that each centroid is in a line of its own.
    - Case of 'Jacobi': The first line will be the eigenvalues, second line onward will be the corresponding eigenvectors (printed as columns).
    - Else: output the required matrix separated by a comma, such that each row is in a line of its own.

Example:

```
>>>python3 spkmeans.py 3 spk input_1.txt
0,4,22
-4.2435,9.1568,5.4105
3.3226,-1.3896,-9.1927
8.2239,-8.5714,-8.4985
```

2.2 C Program (spkmeans.c)

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

1. Reading user CMD arguments:

(a) goal: Can get the following values:

i. wam: Calculate and output the Weighted Adjacency Matrix as described in 1.1.1.

ii. ddg: Calculate and output the Diagonal Degree Matrix as described in 1.1.2.

iii. gl: Calculate and output the Graph Laplacian as described in 1.1.3.

iv. jacobi: Calculate and output the eigenvalues and eigenvectors as described in 1.2.1.

(b) file_name (.txt): The path to the Input file, it will contain **N data points for all above goals except Jacobi, in case the goal is Jacobi the input is a symmetric matrix**, the file extension is .txt

2. Outputting the following

- Case of 'Jacobi': The first line will be the eigenvalues, second line onward will be the corresponding eigenvectors (printed as columns).
- Else: 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:

```
>>>./spkmeans gl input_1.txt
4.2435,9.1568,5.4105
9.1568,1.3896,-8.5714
5.4105,-8.5714,8.4985
```

2.3 Python C API (spkmeansmodule.c)

Start the file with:

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

In this file you will define your C extension which will serve the functions:spk, wam, ddg, gl, jacobi for Python, see 3.

2.4 C Header file (spkmeans.h)

This header have to define all functions prototypes that is being used in spkmeansmodule.c and implemented at spkmeans.c.

2.5 Setup (setup.py)

This is the build used to create the *.so file that will allow spkmeans.py to import mykmeanssp.

2.6 Makefile

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

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

2.7 Build and Running

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

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

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

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

2.8 Assumptions

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

1. No need to validate arguments.

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

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

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

4. Handle errors as following:

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

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

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

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

8. Although all eigenvalues are non-negative in this project, there might be edge cases where you get -0.000 value, this is due to floating point precision, treat it as zero and multiply the corresponding eigenvector by -1.

9. For Kmeans convergence, use $\epsilon = 0,\ max\_iter = 300$.

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) `spkmeans.py`

    (b) `spkmeans.c`

    (c) `spkmeansmodule.c`

    (d) `spkmeans.h`

    (e) `setup.py`

    (f) `Makefile`

    (g) more `*.c, *.h` (optional)

3. Zip the folder using the following Linux cmd:

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

References

[1] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.