Spectral Clustering
Final Project

SUBMISSION DUE DATE: 26/04/2021 23:00
(FINAL DEADLINE! NO EXTENSIONS!)

1 Introduction

In this project you will implement a version of the normalized spectral clustering algorithm. It will include integration of previous homework, generation of data, visualization and implementing a well known clustering algorithm with all of its components.

Spectral clustering, in its many variations, has gained popularity in recent years. Due to its capability of high-quality clustering and handling non-convex clusters that are typically challenging for other methods, spectral clustering has been implemented in different domains like computer vision, bioinformatics and NLP tasks with promising results.

This document will first describe the mathematical basis and algorithms for your project, and then describe the code requirements and implementation. There will not be many implementation details, and no tester is provided – implementation and correctness are up to you. You will be graded for code modularity, design, readability, and performance.

2 Background and Motivation

Given a set of n points $X = \{x_1, x_2, \ldots, x_n\}$ in $\mathbb{R}^d$ we aim at splitting them into a set of k clusters $C = \{C_1, C_2, \ldots, C_k\}$ such that each point $x_i \in X$ is assigned to only one cluster $C_j \in C$.
To make this clustering sensible, we would need a measure to evaluate the similarity of cluster members, and we would aspire to have similar points grouped together.

One way of achieving that is to transform the points into a weighted, undirected graph representation and find the best partition of it into k components.

3 Graph Transformation

As stated before, 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

$V = \{v_1, v_2, \ldots, v_n\}$. 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 $\{v_i, v_j\}$. To represent the weights of the edges and the structure of the graph we introduce the weighted adjacency matrix.

3.1 The Weighted Adjacency Matrix

Let w_{ij} represent the weight of the connection between v_i and v_j. Only if $w_{ij} > 0$, we will define an edge between v_i and v_j.
The weighted adjacency matrix $W \in \mathbb{R}^{n \times n}$ (also referred to as the affinity matrix) is defined as $W = (w_{ij})_{i,j=1,\ldots,n}$.
It is a symmetric ($w_{ij} = w_{ji}$), non-negative ($w_{ij} \geq 0$) representation of weights of all pairs. As we do not allow self loops, we define $w_{ii} = 0$ to avoid them.
In order to create a fully connected graph, the rest of the values are set to:

$$w_{ij} = exp(-\frac{\|x_i - x_j\|^2}{2})$$

We denote $\|\cdot\|^2$ as the ℓ_2-norm or the Euclidean norm.
Once we have defined W, we can present the diagonal degree matrix.

3.2 The Diagonal Degree Matrix

The diagonal degree matrix $D \in \mathbb{R}^{n \times n}$ is defined as $D = (d_{ij})_{i,j=1,\ldots,n}$, such that:

$$d_{ij} = \begin{cases} \sum_{z=1}^{n} w_{iz}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

We get that i-th element along the diagonal equals to the sum of the i-th row of W. In essence, D's diagonal elements represent the sum of weights that lead to vertex v_i. Note that

$$D^{-\frac{1}{2}} = \begin{pmatrix} \frac{1}{\sqrt{d_{11}}} & 0 & \ldots & 0 \\ 0 & \frac{1}{\sqrt{d_{22}}} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \frac{1}{\sqrt{d_{nn}}} \end{pmatrix}$$

3.3 The Normalized Graph Laplacian

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

$$L_{norm} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

The reason we are interested in L_{norm}, is that it has two particularly important properties:

- It has n non-negative real-valued eigenvalues $0 = \lambda_1 \leq \ldots \leq \lambda_n$.
- It has orthogonal real-valued eigenvectors.

4 Eigenvalues and Eigenvectors

In this section, we will lay the foundation needed to find **all** of the eigenvectors and eigenvalues of a real, symmetric, full rank matrix.
Then we will see a heuristic that utilizes the eigenvalues to determine the number of k clusters the data holds. It is a mathematical approach that is very similar to the visual "elbow" method presented in the second homework assignment.

4.1 The Modified Gram-Schmidt Algorithm

The QR decomposition, decomposes a matrix $A \in \mathbb{R}^{n \times n}$ into a product $A = QR$ of an orthogonal matrix Q (i.e. $Q^T Q = I$) and an upper triangular matrix R (i.e. entries below the main diagonal are zero).
One way to compute the QR decomposition is by the Modified Gram-Schmidt algorithm, with the vectors to be considered in the process as columns of the matrix A.
That is, given the matrix $A \in \mathbb{R}^{n \times n}$ and using the notation of B_j to indicate column j of matrix B, and B_{ij} to indicate the element in row i and column j, the algorithm is as follows:

Algorithm 1 The Modified Gram-Schmidt Algorithm

```
1: $U = A$
2: for $i = 1, 2, \ldots, n$ do
3:     $R_{ii} = \|U_i\|^2$
4:     $Q_i = \frac{U_i}{R_{ii}}$
5:     for $j = i + 1, \ldots, n$ do
6:         $R_{ij} = Q_i^T U_j$
7:         $U_j = U_j - R_{ij} Q_i$
8:     end for
9: end for
10: return $Q, R$
```

4.2 The QR Iteration Algorithm

The QR Iteration algorithm outputs an orthogonal matrix $\overline{Q}$ whose columns $\overline{Q}_i$ approach the eigenvectors of A, and a diagonal matrix $\overline{A}$ whose diagonal elements approach the eigenvalues of A. Each eigenvalue $\overline{A}_{jj}$ corresponds to an eigenvector $\overline{Q}_j$.
Thus the QR method presents a simple, elegant algorithm for finding the eigenvectors and eigenvalues of a real, symmetric, full rank matrix.
Given the matrix $A \in \mathbb{R}^{n \times n}$ the QR Iteration algorithm is presented in 2.
As can be seen in step 3 of the algorithm, the upper limit for the iterations is n. This is an approximation we will use, where the "pure" method continues until convergence.
In step 6 we check to see if another iteration is needed or not according to the ϵ difference between the absolute values of each element in $\overline{Q}$ and $\overline{Q}Q$.

Algorithm 2 The QR Iteration Algorithm

```
$\overline{A} = A$
$\overline{Q} = I$
for $i = 1, \ldots, n$ do
    Obtain $Q, R$ for $\overline{A}$ from the Modified Gram-Schmidt algorithm
    $\overline{A} = RQ$
    if $|\overline{Q}| - |\overline{Q}Q| \in [-\epsilon, \epsilon]$ then
        return $\overline{A}, \overline{Q}$
    end if
    $\overline{Q} = \overline{Q}Q$
end for
return $\overline{A}, \overline{Q}$
```

4.3 The Eigengap Heuristic

Determining the number of clusters is a challenging problem and our solution thus far was either prior knowledge (i.e. providing k as input) or a human based visualization method named the "elbow" method.
In the case of spectral clustering, a commonly used heuristic is the eigengap that measures the stability of the eigenvectors in L_{norm} based on its eigenvalues.
Let $(\delta_i)_{i=1,\ldots,n-1}$ be the eigengap for the increasingly ordered eigenvalues $0 \leq \lambda_1 \leq \ldots \leq \lambda_n$ of L_{norm}, defined as:

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

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

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

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

5 Normalized Spectral Clustering

Now, that we have established all of the needed components, we can present the Normalized Spectral Clustering algorithm based on [1,2].

Given a set of n points $X = \{x_1, x_2, \ldots, x_n\}$ in $\mathbb{R}^d$ the algorithm is:

Algorithm 3 The Normalized Spectral Clustering Algorithm

```
1: Form the weighted adjacency matrix $W$ from $X$
2: Compute the normalized graph Laplacian $L_{norm}$
3: Determine $k$ and obtain the first $k$ eigenvectors $u_1, \ldots, u_k$ of $L_{norm}$
4: Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, \ldots, u_k$ as columns
5: Form the matrix $T \in \mathbb{R}^{n \times k}$ from $U$ by renormalizing each of $U$'s rows to have unit length,
   that is set $t_{ij} = u_{ij} / (\sum_j u_{ij}^2)^{1/2}$
6: Treating each row of $T$ as a point in $\mathbb{R}^k$, cluster them into $k$ clusters via the K-means
   algorithm
7: Assign the original point $x_i$ to cluster $j$ if and only if row $i$ of the matrix $T$ was assigned
   to cluster $j$
```

A few things to keep in mind:

- Regarding step 3 - eigenvalues will always be ordered increasingly, respecting multiplicities (for example, all the first 5 eigenvalues are equal to 0). By "the first k eigenvectors" we refer to the eigenvectors corresponding to the k smallest eigenvalues. Determining k would be based on the eigengap heuristic or given as an input (see details in the Implementation section).

- Regarding step 6 - you should use the K-means implementation from the homework assignments, with the needed adjustments to fit your project. You are expected to use the K-means++ initialization when applying K-means. The `MAX_ITER` variable should be set to 300.

6 Implementation

The project will flow along these lines (see later sections for an in-depth breakdown):

1. The user executes the program with certain arguments using the `invoke` library.
2. The program will generate random points that will be used for clustering.
3. The program will compute the clusters using two algorithms: the Normalized Spectral Clustering and K-means.
4. The program will output:
   - A file with the resulting clusters from both algorithms.
   - A file containing the data points that were used.
   - A visualization file comparing the resulting clusters of the two algorithms.

Your main program should be named `main.py` and it will act as the glue for your entire project.

All of the algorithms presented in this assignment **must** be implemented from scratch, i.e. you cannot use external libraries to do it. It is up to you on how to design and execute the architecture of the project. This means deciding on the number of Python extension\s (based on the C API) and the header files and C files that construct them.
This also means, which programming language you use (for example, should you port your K-means++ `Numpy` implementation to C or not).
As for the rest of the project, you can include any libraries you wish, keeping in mind that they must be available on `nova`.

6.1 Coding and Compilation

Your code should be gracefully partitioned into modules (in modules we mean C header files, C API Python extensions, Python modules) and functions. The design of the program, including the main interface\integration, function declarations, partition into modules, and how they interact is entirely up to you, and is graded. You should aim for modularity, reuse of code, clarity, and logical partition.

Modules should contain a main comment that describes the module, its purpose, and interface. Source files should be commented at critical points in the code and at function declarations. Please avoid long lines of code, and shorten lines that exceed 80 characters. Functions should not exceed roughly 60 lines of code.

In your implementation, pay careful attention to the use of constant values and proper use of memory. Do not forget to free any memory you allocated. You should especially aim to allocate only necessary memory and free objects (memory and files) as soon as possible.

In cases where you could have done differently, yet chose a certain way to solve that issue, document your reasoning in the code.

It will include an `invoke` command named `run` (defined in your `tasks.py` file) that will create all of the needed setup to run your program and will execute it (that includes, if needed, calling any `makefile` related commands, building the C API extension\s etc. and executing `main.py`).
To be clear, your project should execute using the following command on `nova`:

```
python3.8.5 -m invoke run -k -n --[no-]Random
```

Where:

- `k` is the number of clusters (or centers).
- `n` is the number of data points.
- `Random` is a `Boolean` typed variable with default value of `True` that indicates the way the data is to be generated.

In this `invoke` command, `k` and `n` could be of type `String` if you decide to.

6.2 Command-Line Arguments

The three command-line arguments of `main.py` will be transferred from the `invoke` command named `run` and will have the same names (see section Coding and Compilation for more details).
In `main.py`, they will be evaluated and checked accordingly (for example, that they have the right typing). In case of errors refer to section Error Handling.
The role of each command-line argument is addressed in the following sections when relevant.

6.3 Random Data Points Generation

Some of the methods in this assignment mainly work in well-separated feature spaces. Coupled with an expensive computation routines and the efficiency of your implementation, can result in run-times that are above 5 minutes.
To address these issues, we define a run-time of 4 minutes and 59 seconds with the maximum amount of points and centers your project can process as *maximum capacity*.

Your program will generate n points that are either 2-dimensional or 3-dimensional with K centers using the `sklearn.datasets.make_blobs` API or an equivalent C library.

The choice between 2 dimension or 3 dimension will be random.
Then, if the `Random` command-line argument is set to `False`, use the supplied command-line arguments `k` and `n` for the values of K and n respectively.
Otherwise (`Random` is set to `True`), the value of K and n would be drawn randomly from the range of *maximum capacity* of your project and half of that value.

For example, if your project can run at *maximum capacity* of $K = 10$ and $n = 200$ for 3-dimensional data points, then for a 3-dimensional data you will randomly sample K to be between 5 and 10 and n to be between 100 and 200.
These bounds should be calculated by you, using trial and error.

At the beginning of the program an informative message should state the *maximum capacity* of the project, meaning, how many points and clusters for both 2-dimensional and 3-dimensional data can it process in a time of less than 5 minutes.
You should allow the user to input values exceeding this value.

6.4 K-means and Normalized Spectral Clustering Comparison

Once we have generated the data we now turn to compute its clusters in two ways. The clusters will result from two algorithms and will be compared. These are the Normalized Spectral Clustering and K-means (with K-means++ initialization).
Note that in this context, the K-means will run as a standalone algorithm (separate instance) on the randomly generated data and has nothing to do with the other K-means algorithm used by the Normalized Spectral Clustering algorithm.
Its respective parameters k, n, d should be set accordingly and with `MAX_ITER` set to 300.

In order to create a fair comparison, both algorithms should use the same data and parameters, in particular, the number of clusters they are expected to calculate k.
If `Random` is set to `True`, k will be computed using the eigengap heuristic and used for them both. Otherwise, both will set k to be K.

6.5 Textual Output Files

The textual output files will consist of two files named `data.txt` and `clusters.txt`. They could be used to analyse and reproduce your results.
The files will be in the following format:

- `data.txt` will contain the generated data from Random Data Points Generation. Each point will have its dimension values separated by commas, where each line represents a single point. The last value of each point will be the integer label for cluster membership of that data point (which you received form the `sklearn.datasets.make_blobs` API).

- `clusters.txt` will contain the computed clusters from both algorithms. The first value will be the number of clusters k that was used. Then, the next k lines will contain in each line the indices of points belonging to the same cluster computed by the Normalized Spectral Clustering algorithm. Finally, the last k lines will have the same format, but this time of the resulting K-means clusters. When using the indices we refer to the first point in `data.txt` as having the index 0, the second as 1 and so on.

For example, given 3-dimensional 10 data points with 3 cluster memberships (were generated from 3 centers), the `data.txt` file would look like this:

```
1.35205173,-2.48226198,-5.81373788,1
1.0952821,-3.23676944,-6.97625046,1
7.1127088,9.39365976,0.39837135,1
7.34199564,5.6401704,1.16838631,2
1.35511932,-3.80569382,-8.6048377,0
6.71733595,8.29076033,2.36514394,2
-3.7727363,-4.77317349,2.19877796,0
1.53174006,-2.08325007,-7.44306108,2
-2.56154974,-2.82810149,4.41503124,0
7.00984847,10.14507189,0.54409154,1
```

And the matching `clusters.txt` file would have this format:

```
3
0,1,2,9
3,5,7
4,6,8
0,1,2
4,5,7,9
3,6,8
```

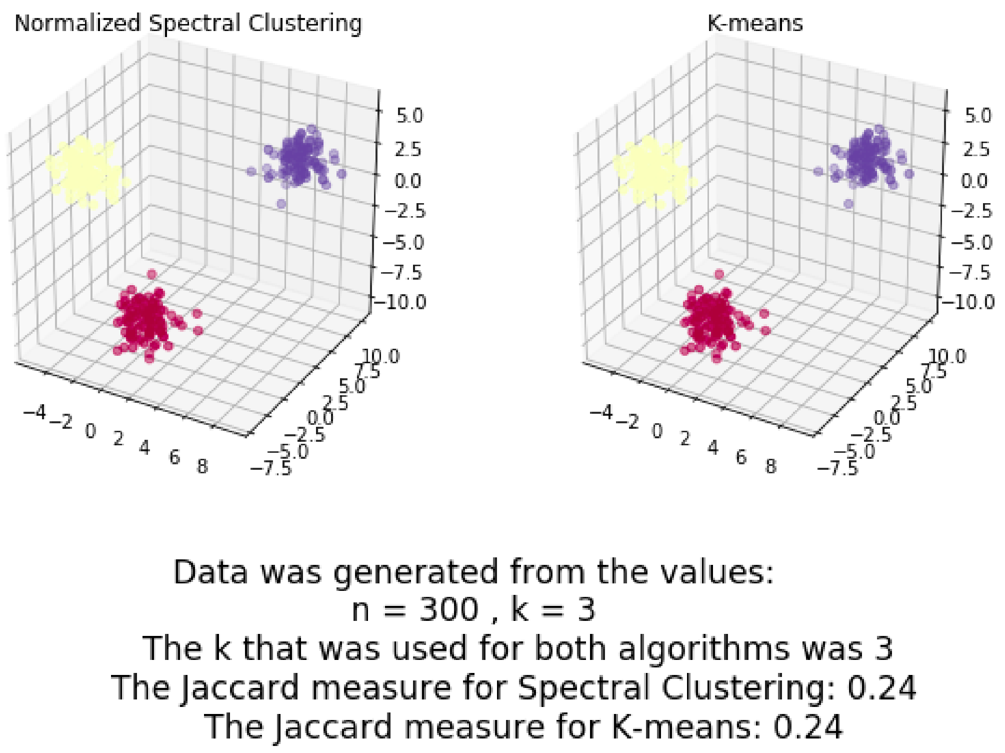
6.6 Visualization File Output

The graphic output of your project should be a `PDF` file named `clusters.pdf`, containing the visualization and information on the clusters you have calculated.
The output will consist of the following:

- **Clustering visualization** - depending on the dimension of the data used, the plot will be either 2-dimensional (with X, Y axis) or 3-dimensional (with X, Y, Z axis). Each calculated cluster, resulting from the Normalized Spectral Clustering algorithm and the K-means algorithm, will be colored differently.

- **Descriptive information** - It will note the `k` and `n` it was generated from (based on Random Data Points Generation) and the k it used for both algorithms. In addition, it will calculate the *Jaccard measure* for each clustering algorithm to display the quality of their clustering.
  The *Jaccard measure* is defined as the number of pairs that are in the same cluster in both the generated (our truth standard) dataset and computed solution divided by the total number of clustered pairs in either solution.

In Figure 1, an example of a 3-dimensional plot is presented. In it, the 3-dimensional data originated from $n = 300$ and $k = 3$ and both clustering algorithms present 3 calculated clusters in different colors. The Jaccard measure is shown as well.



Data was generated from the values:
n = 300 , k = 3
The k that was used for both algorithms was 3
The Jaccard measure for Spectral Clustering: 0.24
The Jaccard measure for K-means: 0.24

Figure 1: An example of a 3-dimensional plot output of `clusters.pdf`. Mock data was used to clarify the visualization requirement.

6.7 Checking Zeros

Due to numerical representations of real values, we will use $\epsilon = 0.0001$ as zero. One such example was introduced in the QR Iteration algorithm.

6.8 Performance Considerations

There can be various bottlenecks in your code, such as the calculations in the search for eigenvectors and eigenvalues.

You are not given a tester, but are instead expected to analyze your code and optimize it by finding these bottlenecks. Store values to avoid repeated calculations, try to access contingent memory, minimize memory allocations, avoid repeated array indexing, etc.
You may freely compare your performance (i.e. your *maximum capacity*) with other students, so long as you do not share anything else.

6.9 Error Handling

Your code should handle all possible errors that may occur (e.g., memory allocation).

Since you are dealing with numeric procedures, numeric problems are to be expected. You should protect your code from problems such as division by zero, infinite loops, etc. You may exit with an error message on each case which leads to it.

Unlike the previous assignments, you may **not** use `assert`. On any error, print a descriptive error message and handle all that needed on your end exit correctly.

6.10 Submission

The project will be submitted by Moodle. The grading may also involve testing of the code in a frontal meeting, to be set as necessary.
Both students should be familiar with a significant part of the code.
The zipped file must contain **all** of the files needed to run your project on `nova`. All files must be in the root, **no folders**.
Both students should submit a zip file named `id1_id2_project.zip`, replacing `id1` and `id2` with the actual 9-digit ID of both partners.
Submission in pairs is mandatory! If you wish to submit alone, contact the TA with a valid reason to get approval. If you didn't find a partner do not contact the TA before posting on the "finding partners" Moodle forum.

MAC users: create (or check) your zip files under Linux, as otherwise hidden auxiliary files are automatically added and will reduce your grade.

6.11 Remarks

- For questions regarding the project, please post in the forum.
- The forum is an integral part of the project. Subscribe to it and follow its messages.
- Borrowing from others' work is unacceptable and bears severe consequences.
- The submission date and time is final! Late submissions are not allowed and will not be approved. There will be no extensions under any circumstances, except miluim or hospitalization. Sick notes do not allow for extensions. You are expected to complete and submit the project at least two weeks ahead of the submission date, and failure to do so is your responsibility. Any submission beyond the deadline will result in a 10 points deduction per day (or part of it), until the submission closes. Check Moodle for the exact deadline and closing time of the submission.

GOOD LUCK!

References

[1] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 14:849–856, 2001.

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