University of Edinburgh
INFR11156: Algorithmic Foundations of Data Science (2019)
Coursework

Due: 4pm, 8 November, 2019

Submission instructions

Your submission should be contained in a directory named as your uun. Follow the following directory structure for your submission:

- Part A: A PDF document containing your answers. Typeset with LaTex is highly recommended. A LaTex template for writing the coursework solutions is provided and can be downloaded from the course webpage from Learn.
- Part B:
  - readme.txt (max 1 page) - Instructions for running your code and any comments or interesting observation you want us to know.
  - source - contains all the source files
  - build - compiled binary (e.g., jar file) (optional)
  - dependency - contains any external libraries you use. E.g., external Jar files.

  Finally, make a .tar.gz file named as <uun>.tar.gz and submit using the submit command on DICE:

```
submit afds cw1 filename
```

University regulations

University Regulations On Good Scholarly Practice. Please remember the University requirement as regards all assessed work. Details about this can be found at:

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Remember, if you use ideas from elsewhere (including other students), cite them. And try not use too much of these. The regulation says you can pick up "general ideas" but not "pivotal ideas". But "general" and "pivotal" are subjective. Play safe and avoid getting into trouble.

Part A: Theoretical part (50%)

We have seen in class that a random vector $x \in \{-1, 1\}^n$ can be used as an initial vector for the Power Method to approximate the largest eigenvalue of a PSD matrix. In this question, you need to prove or disprove that a random vector $x \in \mathbb{R}^n$, where every x_i is generated according to $\mathcal{N}(0, 1)$, can be used for the same purpose. That is, you need to prove or disprove that, for a suitable choice of k, with constant probability the output vector x_k from Algorithm 1 below satisfies

$$\frac{\mathbf{x}_k^\intercal B \mathbf{x}_k}{\mathbf{x}_k^\intercal \mathbf{x}_k} \geq (1 - \epsilon) \cdot \frac{\lambda_1}{1 + 4n(1 - \epsilon)^{2k}},$$

where λ_1 is the largest eigenvalue of matrix B. The following lemma might be used in your analysis.

Lemma. Let Y be a random variable generated according to a χ^2-distribution with n degrees of freedom. Then, it holds that $\mathbf{P}[Y > n] < 1/2$.

Algorithm 1: Power Method with Normal Distribution

Input: A PSD symmetric matrix $B \in \mathbb{R}^{n \times n}$, and $k \in \mathbb{Z}^+$.

Let $\mathbf{x}_0 \in \mathbb{R}^n$, where each coordinate x_i is sampled independently from $\mathcal{N}(0, 1)$
for $i = 1$ *to* k **do**
  $\mathbf{x}_i = B\mathbf{x}_{i-1}$
end
return $\mathbf{x}_k$

Part B: Programming part (50%)

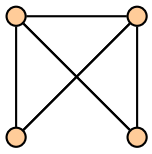
In this question, you will apply the Power Method and a simple algorithm to segment an image into two regions using a graph partitioning algorithm. In particular, you will see that how the Power Method could be used to identify objects from images like the figure shown on the right.

Background Reading. The first task behind our job is to model an input image as a mathematical object. Here, we choose to represent an image as an undirected and weighted graph $G = (V, E, w)$ with weight function $w : E \to \mathbb{R}_{\geq 0}$ constructed as follows: each pixel corresponds to a vertex $u \in V[G]$. Each pixel is also represented as a point (r, g, b, x, y) where $r, g, b \in [0, 1]$ are the normalised RGB values of the pixel and $x, y \in [0, 1]$ are the normalised coordinates of the pixel in the image. For any two pixels x and y, their corresponding vertices in G are connected by an edge with weight

$$\exp\Big(-4\|\mathbf{x} - \mathbf{y}\|^2\Big),$$

where $\mathbf{x}$ and $\mathbf{y}$ are the points corresponding to pixels x and y. For efficiency, you should only include edges between pixels close to each other; for example, ± 5 rows and columns. Additionally, you should ignore edges with weights below some threshold t. Suggested parameter is $t = 0.9$. Please note that you should take the "wrap around" effect of the pixels into account, i.e., the vertices corresponding to pixels near the top (or left) of the image should be connected to the vertices corresponding to pixels near the bottom (or right) and the distance between their (x, y) coordinates should be considered small.

As the second step, we use the following so-called *normalised Laplacian matrix* $\mathcal{L}_G$ to represent G. The normalised Laplacian matrix $\mathcal{L}_G \in \mathbb{R}^{n\times n}$ is defined as follows: (1) All the diagonal entries satisfy $(L_G)_{u,u} = 1$; (2) For any edge $\{u, v\} \in E(G)$, $(L_G)_{u,v} = (L_G)_{v,u} = -1/\sqrt{d_u \cdot d_v}$, where $d_u = \sum_z w(u, z)$ and d_v is defined in the same way; (3) For any other entries indexed by u, v, $(L_G)_{u,v} = 0$. The matrix $\mathcal{L}_G$ can also be defined as $I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ where A is the adjacency matrix of the graph and D is the diagonal matrix with $D_{u,u} = d_u$. The figure below gives an example of a graph and its corresponding normalised Laplacian matrix.



$$\mathcal{L}_G = \begin{pmatrix} 1 & 0 & -1/\sqrt{6} & -1/\sqrt{6} \\ 0 & 1 & -1/\sqrt{6} & -1/\sqrt{6} \\ -1/\sqrt{6} & -1/\sqrt{6} & 1 & -1/3 \\ -1/\sqrt{6} & -1/\sqrt{6} & -1/3 & 1 \end{pmatrix}$$

Figure 1: The normalised Laplacian matrix of a graph.

In the third step, the following Algorithm 2 uses the eigenvector associated with the second smallest eigenvalue of $\mathcal{L}_G$ to segment an image into two regions. Here, the function $\Phi(S)$ used in the pseudocode is defined by

$$\Phi(S) = \frac{w(S, V \setminus S)}{\min(\mathrm{vol}(S), \mathrm{vol}(V \setminus S))},$$

where

$$w(S, V \setminus S) = \sum_{\substack{\{u,v\}\in E(G) \\ u\in S, v\in V\setminus S}} w(u, v),$$

and $\mathrm{vol}(S) = \sum_{u\in S} d_u$.

Algorithm 2: Algorithm for finding a sparse cut

```
f_2 = eigenvector of L_G corresponding to λ_2;
Sort the vertices of graph u_1, u_2, ..., u_n such that f_2(u_1) ≤ ... ≤ f_2(u_n);
Let t be a parameter, and t = 0 initially;
Let S be a set of vertices, and S = ∅ initially;
S* = {u_1}
while t < n do
    t = t + 1
    S = S ∪ {u_t}
    if Φ(S) < Φ(S*) then
        S* = S
    end
end
```

Question 1: Write a function which reads a PNG image file and downsamples it to create a 100×100 image. You should also smooth the downsampled image slightly, for example by applying Gaussian blur. The function should display the downsampled image.

Question 2: Write a function which takes the output from Question 1 and constructs the adjacency matrix of a graph representing a given image. The function should print the number of edges and average degree of the constructed graph.

Question 3: Write a function which takes the output from Question 2 and uses the Power Method to compute the vector corresponding to the second smallest eigenvalue of $\mathcal{L}_G$. You are free to choose the parameter k which controls the number of iterations in the Power Method.

Note that the Power Method finds the largest eigenvalues, so you should apply the Power Method to $2I - \mathcal{L}_G = I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. You may use the fact that the eigenvector with the largest eigenvalue is $D^{\frac{1}{2}}\mathbf{1}$ where $\mathbf{1}$ is the constant vector.

The function should print your chosen value of k and display an image showing the value of the computed vector at each pixel in the downsampled image.

Please note that you may not use a library for computing eigenvalues or eigenvectors in this question.

Question 4: Write a function which uses the output of Question 3 and Algorithm 2 to find a sparse cut S, in the graph corresponding to an image. The function should display an image showing which pixels of the downsampled image are in the set S.

Example. Figure 2 shows a sequence of expected results for each question, when the image on Page 2 is given as input. Please note that a marker will use their "common sense" to judge the quality of your image segmentation result. For instance, you will get 0 points for this instance if your output is like Figure 3.

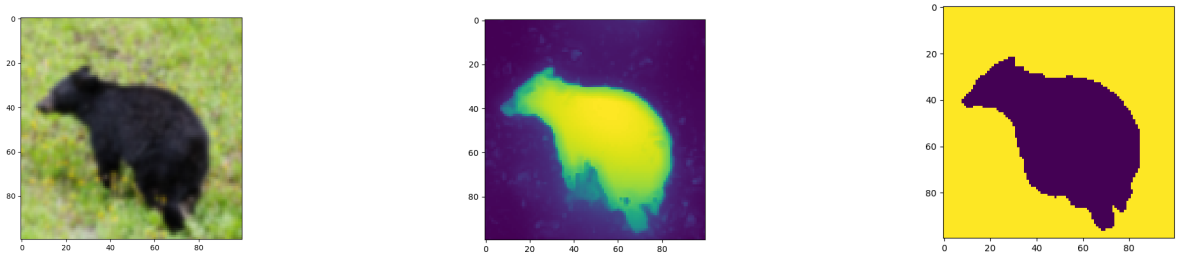


Figure 2: From left to right: the expected output of the segmentation algorithm after Questions 1, 3, and 4.

Marking Criteria. The testset will consist of 5 images. Your algorithm's performance on each image is worth 10% of the coursework mark. This is split across the 4 questions as follows.

- Question 1: 2%
- Question 2: 2%
- Question 3: 3%
- Question 4: 3%

Please notice that, if your algorithm does not work for some question, then you will not receive any marks for the subsequent questions.

Programming Guidelines. You may use any programming language for this question as long as your code runs on a DICE machine. You must provide a `readme.txt` file with your submission with instructions for how to run your code. The time limit for each input instance is 10 seconds. The

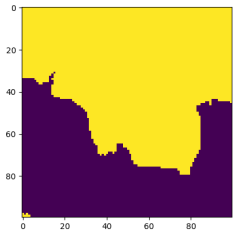


Figure 3: An example of poor output from the segmentation algorithm.

background reading section recommends some choices of the parameters. However, you could choose different parameters to improve the runtime and the quality of the output.

Tips

1. Use a sparse matrix data structure for the graph matrices. For example, see `scipy.sparse` for Python.
2. Use standard libraries for matrix-vector multiplication rather than for loops wherever possible.
3. Three images are provided for testing your implementation. Your submission will be evaluated with similar images.