National Technical University of Athens

Interdisciplinary Master's Programme in

Data Science and Machine Learning

ALGORITHMIC DATA SCIENCE

Exercise Sheet 2

[redacted]

July 4, 2022

Contents

1 Hashing & Universality

Let U be a non empty set. A family of hash functions $\mathscr{H} = \{h : U \to [m]\}$ is called *universal* if

$$\forall x, y \in U, x \neq y : \mathbb{P}_{h \in \mathscr{H}}[h(x) = h(y)] \leq \frac{1}{m}. \tag{1.1}$$

Equivalently, if for every two distinct values $x, y \in U$ there exists at most $|\mathscr{H}|/m$[1] functions $h \in \mathscr{H}$ for which $h(x) = h(y)$.

1.1 Solution to 1.A

Prove that for $\alpha \in [m] \setminus \{0\}$, $b \in [m]$ the family of functions $h_{\alpha,\beta}(x) = (\alpha x + \beta) \pmod m$ is not universal when $U = [m^k]$, $k \geq 2$.

Proof. First observe that we can rewrite the probability in (1.1) as

$$\begin{aligned} \mathbb{P}_{h \in \mathscr{H}}[h(x) = h(y)] \leq \frac{1}{m} &\iff \frac{|\{h \in \mathscr{H} : h(x) = h(y)|\}}{|\mathscr{H}|} \leq \frac{1}{m} \\ &\iff |\{h \in \mathscr{H} : h(x) = h(y)| \leq \frac{|\mathscr{H}|}{m}. \end{aligned} \tag{1.2}$$

Therefore, to prove that $\mathscr{H}$ is not universal it suffices to show that there exists a pair of points $x \neq y,\ x, y \in U$ for which

$$|\{h \in \mathscr{H} : h(x) = h(y)| > \frac{|\mathscr{H}|}{m}. \tag{1.3}$$

To this end, we first prove that $|\mathscr{H}| = m(m-1)$. First observe that $\mathscr{H}$ has at most $m(m-1)$ elements. Now, suppose for the sake of contradiction we have two pairs $(\alpha, \beta), (c, d) \in [m] \setminus \{0\} \times [m]$ for which $h_{\alpha,\beta}(x) = h_{c,d}(x)$ for all $x \in U$. In particular, for $x = 0$ we obtain

$$\begin{aligned} h_{\alpha,\beta}(0) = h_{c,d}(0) &\iff \beta = c \pmod m \\ &\iff m \mid \beta - c, \end{aligned}$$

which implies that $\beta = c$, since both $\beta, c \in [m]$. Similarly, for $x = 1$ we have that

$$\begin{aligned} h_{\alpha,\beta}(1) = h_{c,d}(1) &\iff \alpha + \cancel{\beta} = c + \cancel{d} \pmod m \\ &\iff \alpha = c \pmod m, \end{aligned}$$

which implies that $\alpha = c$, since $\alpha, c \in [m] \setminus \{0\}$. Therefore, we showed that for each pair $(\alpha, \beta) \in [m] \setminus \{0\} \times [m]$ the function $h_{\alpha,\beta}$ is unique in $|\mathscr{H}|$. In other words, there exists a one-to-one correspondence $[m] \setminus \{0\} \times [m] \mapsto |\mathscr{H}|$. Since $\mathscr{H}$ has at most $m(m-1)$ elements and both sets are finite we conclude that they have the same cardinality. But, the cardinality of $[m] \setminus \{0\} \times [m]$ is equal to $m(m-1)$. Hence, $|\mathscr{H}| = m(m-1)$. Now, we claim that the magic pair

[1] Where by $|\mathscr{H}|$ we denote the cardinal number of $\mathscr{H}$.

that satisfies (1.2) is $(x,y)=(m,0)$. Indeed, first observe that for every $\alpha \in [m]\setminus\{0\}$, $\beta \in [m]$ we have that

$$\begin{aligned} h_{\alpha,\beta}(0)=h_{\alpha,\beta}(m) &\iff \cancel{\beta}=\alpha m+\cancel{\beta} \pmod m \\ &\iff \alpha m=0 \pmod m \iff m\,|\,\alpha m, \end{aligned}$$

which is trivially true. The latter means that for every pair $(\alpha,\beta)\in[m]\setminus\{0\}\times[m]$ we have that $h_{\alpha,\beta}(0)=h_{\alpha,\beta}(m)$. In other words, every function in $h\in\mathscr{H}$ satisfies $h(0)=h(m)$. Hence,

$$|\{h\in\mathscr{H} : h(0)=h(m)|=m(m-1)>\frac{|\mathscr{H}|}{m}=\frac{m(m-1)}{m}=(m-1),$$

which proves that (1.3) is true for $x=0,y=m$ and therefore $\mathscr{H}$ is not universal. □

1.2 Solution to 1.B

Prove that for every prime number $p>m^k$, $k\geq 2$ and for every $\alpha\in[p]\setminus\{0\}$, $\beta\in[p]$ the family of functions $h_{\alpha,\beta}(x)=((\alpha x+\beta) \pmod p) \pmod m$ is universal when $U=[m^k]$.

Proof. To prove that the family $\mathscr{H}=\{h_{\alpha,\beta} : \alpha\in[p]\setminus\{0\},\beta\in[p]\}$ is universal we let $\alpha\in[p]\setminus\{0\}$, $\beta\in[p]$ and a pair $k>l$, $k,l\in U$. We must count for how many pairs α,β the equation $h_{\alpha,\beta}(k)=h_{\alpha,\beta}(l)$ holds. We proceed by solving $h_{\alpha,\beta}(k)=h_{\alpha,\beta}(l)$. We have that

$$\begin{aligned} h_{\alpha,\beta}(k)=h_{\alpha,\beta}(l) &\iff \underbrace{((\alpha k+\beta) \pmod p)}_{r}\equiv\underbrace{((\alpha l+\beta) \pmod p)}_{s} \mod m \\ &\iff s\equiv r \mod m \iff m\,|\,s-r, \end{aligned}$$

where $s,r\in\{0,\dots,p-1\}$. Now, we suppose without loss of generality that $s-r>0$. Then, $s-r\in\{0,\dots,p-1\}$, since both $s,r\in\{0,\dots,p-1\}$. Therefore, if $m\,|\,(s-r)$ then $r-s=nm$, for some $n\geq 0$. Since, $s-r\in\{0,\dots,p-1\}$ we have that

$$\begin{aligned} 0\leq s-r\leq p-1 &\iff 0\leq nm\leq(p-1) \\ &\iff 0\leq n\leq\frac{p-1}{m}. \end{aligned} \tag{1.4}$$

Therefore, there at most $\left[\frac{p-1}{m}\right]+1$ (including zero), $(s-r)\in\{0,\dots,p-1\}$ for which $m\,|\,s-r$. But now, since p is prime, the element $k-l\neq 0$ has an inverse in $\mathbb{Z}_p$. Therefore,

$$s-r\equiv\alpha(k-l) \mod p \iff (k-r)^{-1}(s-r)\equiv\alpha \mod p. \tag{1.5}$$

(1.5) says that there is a one to one correspondence between the α's and the numbers $e=s-r\in\mathbb{Z}_p$ for which $s>r$ and $m\,|\,s-r$. But from (1.4) these numbers e are at most $[(p-1)/m]+1$.

Now, observe that if $s-r=0$ then $\alpha l+\beta \equiv \alpha k+\beta \mod p \iff p \,|\, \alpha(k-l)$. The fact that p is prime and $\alpha \in \{1,\ldots,p-1\}$ implies that $p \,|\, (k-l)$. The latter implies that $k \equiv l \mod p$, which cannot be true since we have assumed that $k \neq l$ in $\mathbb{Z}_p$. Therefore, we conclude that the α's for which $h_{\alpha,\beta}(k) = h_{\alpha,\beta}(l)$ are at most $\left\lceil \frac{p-1}{m} \right\rceil$. Since this holds for every β independently, we have that the total number of functions $h \in \mathscr{H}$ for which $h(k)=h(l)$ is at most $p\left\lceil \frac{p-1}{m} \right\rceil$. This means that

$$\mathbb{P}_{h\in\mathscr{H}}\left[h(k)=h(l)\right] \leq p\frac{\left\lceil \frac{p-1}{m} \right\rceil}{|\mathscr{H}|} \leq \frac{\cancel{p}\cancel{(p-1)}}{m\cancel{p}\cancel{(p-1)}} = \frac{1}{m},$$

which in turn implies that $\mathscr{H}$ is universal. □

1.3 Solution to 1.C

Is it true that the family $\mathscr{H}$ is still universal if we replace $U=[p]$ instead of $U=[m^k]$ in 1.2?

Answer: The family is still universal even if we replace $U=[p]$ instead of $U=[m^k]$. To see this, observe that the proof provided in 1.2 is still valid since we didn't make use of the fact that $k,l \in [m^k]$ but only that $|k-l| \in \{0,\ldots,p-1\}$ which is still true in the case where $k,l \in [p]$.

2 Hashing and open addressing

2.1 Hash search & Hash insert

The proof given in [1] refers to the expected number of probes in an unsuccessful search and to the expected number of probes of inserting an element. The claim is that these two coincide. Below we give the pseudocode for these two operations.

Algorithm 1 Hash-Insert (T, k)

```
1: $i = 0$
2: repeat
3:     $j = h(k, i)$
4:     if $T[j] == \text{NIL}$ then
5:         $T[j] = k$
6:         return $j$
7:     else
8:         $i = i + 1$
9:     end if
10: until $i == m$
11: Error: "hash table overflow"
```

Algorithm 2 Hash-Search (T, k)

```
1: $i = 0$
2: repeat
3:     $j = h(k, i)$
4:     if $T[j] == k$ then
5:         return $j$
6:     end if
7:     $i = i + 1$
8: until $T[j] == \text{NIL}$ or $i == m$
9: return NIL
```

To prove that the expected number of probes for an unsuccessful search is equal to the expected number of probes for inserting an element into a hash-table with open-addressing and load factor $\alpha = m/n$, we first estimate the average number of probes of the latter. The proof is essentially the same as the one presented in [1].

Theorem 1. *Given an open-address hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$, assuming uniform hashing.*

Proof. In an unsuccessful search, every probe but the last accesses an occupied slot that does not contain the desired key, and the last slot probed is empty. Denote by X the number of probes made in an unsuccessful search and by A_i, for $i = 1,2\ldots$, to be the event that an ith probe occurs and it is an occupied slot. Then, it follows that $\{X \geq i\} = A_1 \cap A_2 \cap \cdots \cap A_{i-1}$. Using the general product rule we may write

$$\begin{aligned}\mathbb{P}(A_1 \cap A_2 \cap \cdots \cap A_{i-1}) = \mathbb{P}(A_1) \cdot \mathbb{P}(A_2 \,|\, A_1) \cdot \mathbb{P}(A_3 \,|\, A_1 \cap A_2) \ldots \\ \mathbb{P}(A_{i-1} \,|\, A_1 \cap A_2 \cap \cdots \cap A_{i-2}).\end{aligned}$$

Since there are n elements and m slots, $\mathbb{P}(A_1) = n/m$. For $j > 1$ the probability that there is a jth probe and it is an occupied slot, given the first $j-1$ probes were to occupied slots, is $(n-j+1)/(m-j+1)$. This probability follows because we would be finding one of the remaining $(n-(j-1))$ elements in one of the $(m-(j-1))$ unexamined slots, and by the assumption of uniform hashing, the probability is the ratio of these quantities. Observing that $n < m$ implies that $(n-j)/(m-j) \leq n/m$ for all j such that $0 \leq j < m$, we have for all i such that $1 \leq i \leq m$,

$$\begin{aligned}\mathbb{P}(X \geq i) &= \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2} \\ &\leq \left(\frac{n}{m}\right)^{i-1} = \alpha^{i-1}. \qquad (2.1)\end{aligned}$$

Now, using (2.1) we have that the expcted number of probes in an usuccessful search is equal to

$$\begin{aligned}\mathbb{E}(X) &= \sum_{i=1}^{\infty} \mathbb{P}(X \geq i) \leq \sum_{i=1}^{\infty} \alpha^{i-1} \\ &= \sum_{i=0}^{\infty} \alpha^{i} = \frac{1}{1-\alpha}.\end{aligned}$$

□

Using Theorem 1 we can prove that the expected number of probes for unsuccessful search is equal to the expected number of probes for inserting an element. Formally, we have the following corollary.

Corollary 1.1. Inserting an element in an open-address hash table with load factor α requires at most $1/(1-\alpha)$ probes on average, assuming uniform hashing.

Proof. An element is inserted only if there is room in the table, and thus $\alpha < 1$. Inserting a key requires an unsuccessful search followed by placing the key into the first empty slot found. Thus, the expected number of probes is at most $1/(1-\alpha)$. □

2.2 Expected number of probes in a successful search

Given an open-address hash table with load factor $\alpha < 1$, the expected number of probes in a successful search is at most $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$.

Proof. The proof is given in [1]. We provide the proof for completion. A search for a key k reproduces the same probe sequence as when element with k was inserted. By 2.1, if k was the $(i+1)$st key inserted into the hash table, the expected number of probes made in a search for k is at most $1/(1-i/m) = m/(m-i)$. Averaging over all n keys in the hash table gives us the expected number of probes in a successful search:

$$\begin{aligned}
\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} = \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} &= \frac{1}{\alpha}\sum_{k=m-n+1}^{m}\frac{1}{k} \\
&\leq \frac{1}{\alpha}\int_{m-n}^{m}\frac{1}{x}\,dx = \frac{1}{\alpha}\ln\frac{m}{m-n} \\
&= \frac{1}{\alpha}\ln\frac{1}{1-\alpha}.
\end{aligned}$$

□

3 The Girvan-Newman Algorithm

3.1 Method and Complexity

Describe in detail, in the form of pseudocode the Girvan-Newman algorithm for the calculation of the edge betweeness in a graph and determine its complexity.

Answer: Before we present the Girvan-Newman algorithm we recall the definition of *betweeness* in an undirected graph $G = (V,E)$. The *betweeness* of an edge (α, β) is defined to be the sum of the fractions of the shortest paths from any pair of nodes $x, y \in V$ that include the edge (α, β) to the total of shortest paths between x, y.

In order to exploit the betweenness of edges, we need to calculate the number of shortest paths going through each edge. The Girvan-Newman algorithm visits each node X once and computes the number of shortest paths from X to each of the other nodes that go through each of the edges. The algorithm can be divided in three steps:

- **Step 1:** For a given node $X \in V$, the algorithm performs a breadth-first search (BFS) and constructs the BFS tree T of the graph. The BFS tree consists of levels; the root X belongs to the zero level. The r-th level of the BFS tree consists of nodes that are r edges "away" from the 0-th level. Edges between levels are called *DAG*[2] edges. Each DAG edge will be part of at least one shortest path from root X. We denote this operation by $T \leftarrow BFS(X)$, where the BFS starts from node X and returns the BFS tree.

- **Step 2:** The second step of the algorithm is to label each node by the number of shortest paths that reach it from the root. Start by labeling the root 1. Then, from the top down, label each node Y by the sum of the labels of its parents. We denote this operation by $Shortest\_paths(T)$ and it is applied to the BFS tree T.

- **Step 3:** The third and final step is to calculate for each edge e the sum over all nodes Y of the fraction of shortest paths from the root X to Y that go through e. This operation proceeds as follows: Each leaf in the BFS tree gets a credit of 1. Each node that is not a leaf gets a credit equal to 1 plus the sum of the credits of the DAG edges from that node to the level below. A DAG edge e entering node Z from the level above is given a share of the credit of Z proportional to the fraction of shortest paths from the root to Z that go through e. Formally, let the parents of Z be $Y_1, Y_2, \ldots, Y_k$. Let p_i be the number of shortest paths from the root to Y_i. Then the credit for the edge (Y_i, Z) is the credit of Z times p_i divided by $\sum_{j=1}^{k} p_j$. We denote this operation by *Calculate_edge_credits(T)*.

After performing the credit calculation with each node as the root, we sum the credits for each edge. Then, since each shortest path will have been discovered twice - once when each of its endpoints is the root - we must divide the credit for each edge by 2.

[2]*DAG* stands for directed acyclic graph.

In Algorithm 3 we summarize the previous steps in the form of pseudocode and we calculate the complexity of the algorithm.

Algorithm 3 Girvan-Newman - Edges betweenness centrality

```
Require: An undirected graph $G = (V, E)$.
1: Set credits to zero; $G.E \leftarrow 0$.
2: for each node $X \in V$ do
3:     $T \leftarrow BFS(X)$.
4:     Apply Shortest_paths(T).
5:     $G.E \leftarrow G.E + Calculate\_edge\_credits(T)$.
6: end for
7: return G.
```

To calculate the complexity of Algorithm 3 we see that the step 3 requires time $O(|V| + |E|)$. Now since the operation for the calculation of the shortest paths is applied to the BFS tree T and not to the complete graph G, we have that the total edges E' of T are equal to $|V| - 1$. The operation requires $O(|V| + |E'|) = O(|V|)$ time. Similarly, the calculation of credits per edge in T requires also $O(|V| + |E'|) = O(|V|)$ time. Now, the for loop in 2 indicates that these operations must be carried out for each node of G. Therefore, the complexity of Algorithm 3 is equal to

$$O(|V|) \cdot O(3|V| + |E|) = O(3|V|^2 + |V| \cdot |E|) = O\left(|V|(|V| + |E|)\right).$$

But since in most cases we have $|E| > |V|$, therefore the time complexity of the Girvan-Newman algorithm in most cases is $O(|V| \cdot |E|)$.

3.2 Edges betweenness in case of trees

How the algorithm can be simplified in case where the graph $G = (V, E)$ is a tree? Determine the complexity for the simplified version.

Answer: In the case where the graph $G = (V, E)$ is a tree then $|E| = |V| - 1$. Moreover, for every pair of nodes (α, β) the exists a unique path that connects α, β. Hence, in order to find the edge betweenness of an edge (α, β) it suffices to find the nodes that lie on the side of α and the nodes on the side of β. More formally, suppose that we remove the edge (α, β) from the complete tree T. Then, T is partioned into two subtrees T_1, T_2. Suppose that $\alpha \in T_1$ and $\beta \in T_2$. Then the nodes that lie on the side of α are contained in T_1 and the nodes that lie on the side of β lie in T_2. Now, the number of shortest paths that pass through the edge (α, β) is equal to $|T_1| \cdot |T_2|$. Therefore, the edges betweenness of (α, β) is $|T_1| \cdot |T_2|$. Since the above procedure must be repeated for every edge, the complexity of the algorithm will be equal to $O(|E|) = O(|V|)$. This complexity is better than of GN's which is $O(|V|^2)$ in the case of trees. More about the simplified version can be found in [3].

4 Cuts based on Spectral Analysis

4.1 Exercise 10.4.1 - MMDS

Construct:

(i) The adjacency matrix,

(ii) The degree matrix,

(iii) The Laplacian matrix,

for the graph given below:

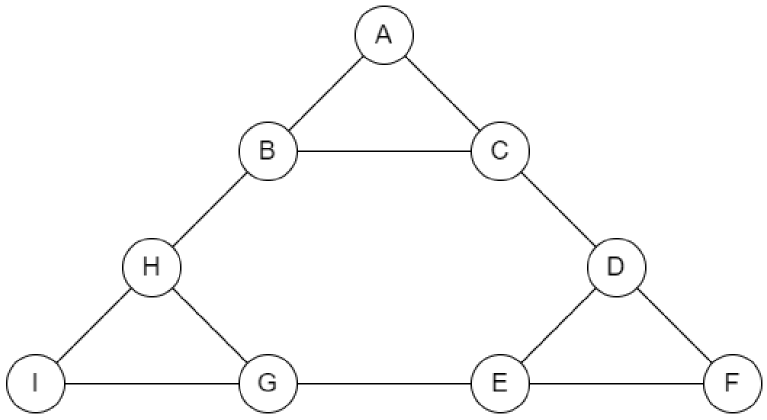

Figure 1: The graph in MMDS.

Solution: (i) To compute the adjacency matrix in Figure 1 we assume the alphabetical order ABCDEFGHI. The adjacency matrix for the given graph is a matrix $A = (\alpha_{ij})$ of dimensions 9×9 where the entries of the matrix are calculated as follows:

$$\alpha_{ij} = \begin{cases} 1, & \text{if there is a connection between the nodes } i, j \\ 0, & \text{otherwise} \end{cases} .$$

Considering the predefined order, the first row of A corresponds to the node A, the second to the node B, etc. Then, in this case the adjacency matrix is given by

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}. \tag{4.1}$$

(ii) Following the same order as in (i), the degree matrix $D = (d_{ij})$ is a diagonal 9×9 diagonal matrix where the element d_{ii} corresponds to the degree of the i-th node in the graph. Therefore, the degree matrix for the graph given in Figure 1 is equal to

$$D = \operatorname{diag}(2,3,3,3,3,2,3,3,2). \tag{4.2}$$

(iii) The Laplacian matrix is defined to be the matrix $L = D - A$, where D is the degree matrix and A is the adjacency matrix. Using (4.1), (4.2) we obtain

$$L = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 3 & -1 & -1 \\ 0 & -1 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix}. \tag{4.3}$$

4.2 Exerice 10.4.2 - MMDS

For the Laplacian matrix in (4.1), find the second-smallest eigenvalue and its eigenvector. What partition of the nodes does it suggest?

Solution: To find the eigenvalues of the Laplacian matrix L we need to find the roots of the characteristic polynomial $\phi(x) = \det(L - xI_9)$, where I_9 denotes the 9×9 identity matrix. The fact that L is symmetric ($L = L^T$) implies that there exists an orthonomal basis of $\mathbb{R}^9$ consisting of eigenvectors of L. Therefore, the characteristic polynomial $\phi(x)$ we will be a product of prime factors. By expanding the determinant with respect to the first column for example, it easily shown that

$$\begin{aligned} \phi(x) &= -x(x-5)(x-3)^3(x^2-5x+3)^2 \\ &= -x(x-5)(x-3)^3\left[\left(x - \frac{5-\sqrt{13}}{2}\right)^2 \cdot \left(x - \frac{5+\sqrt{13}}{2}\right)^2\right]. \end{aligned} \tag{4.4}$$

Therefore, we see that the eigenvalues of L are

$$\lambda_1 = 0, \quad \lambda_2 = \frac{5-\sqrt{13}}{2}, \quad \lambda_3 = 3, \quad \lambda_4 = \frac{5+\sqrt{13}}{2}, \quad \lambda_5 = 5.$$

The second-smallest eigenvalue of L is $\lambda_2 = \frac{5-\sqrt{3}}{2}$ with multiplicity 2. Therefore, there are exactly two eigenvector of L corresponding to λ_2. To obtain the eigenvectors corresponding to

λ_2 we use Python and the library numpy. The two eigenvectors are

$$v_1 = (0.14, -0.02, 0.20, 0.35, 0.19, 0.42, -0.33, -0.4, -0.5)$$

and

$$v_2 = (-0.5, -0.38, -0.38, 0.08, 0.29, 0.29, 0.29, 0.08, 0.29).$$

By looking at the signs of each coordinate in v_1, v_2 we see that $\mathrm{sgn}(v_1) = (+,-,+,+,+,+,-,-,-)$ and $\mathrm{sgn}(v_2) = (-,-,-,+,+,+,+,+,+)$. Therefore the cut proposed by v_1 is $S_1 = \{A,C,D,E,F\}$, $T_1 = \{B,G,H,I\}$ and the cut proposed by v_2 is $S_2 = \{A,B,C\}$, $T_2 = \{D,E,F,G,H,I\}$. In Figure 2 we see the cuts proposed by the eigenvectors v_1, v_2.

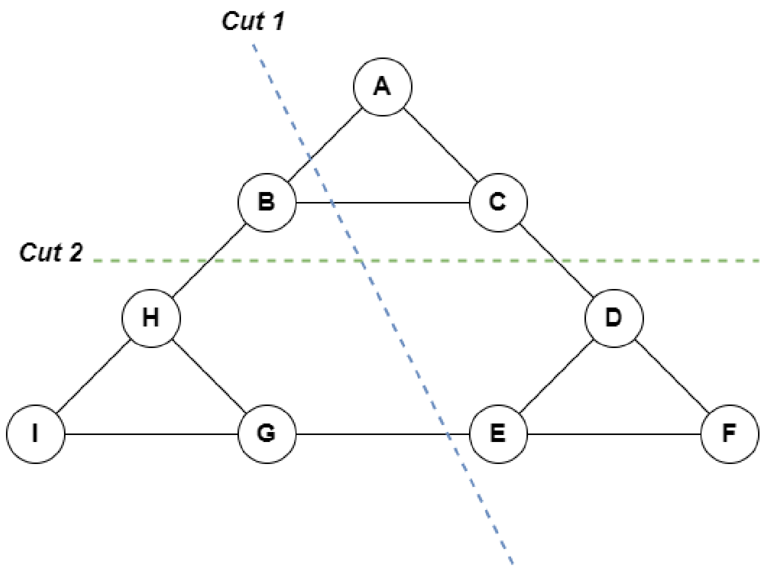

Figure 2: The cuts proposed by the eigenvectors v_1, v_2. Cut 1 (blue-dashed line) corresponds to the cut proposed by v_1. It partitions the vertices of the graph into two disjoint subsets $S_1 = \{A,C,D,E,F\}$, $T_1 = \{B,G,H,I\}$. Cut 2 (green-dashed line) corresponds to the cut proposed by v_2. The partition consists of the sets $S_2 = \{A,B,C\}$ and $T_2 = \{D,F,E,H,G,I\}$.

5 Normalized cuts & Modularity

5.1 Normalized Cut

Compute the normalized cut for the partition $S = \{1, 2, 3\}$, $T = \{4, 5, 6\}$ for the graph given below.

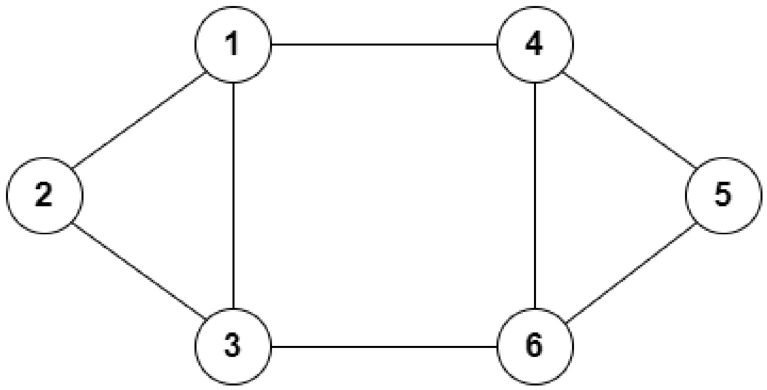

Figure 3: Figure 10.16 in MMDS.

Solution: The *normalized cut* for the partition $S = \{1, 2, 3\}$, $T = \{4, 5, 6\}$ is given by the following formula:

$$\mathrm{NC}(S,T) = \frac{\mathrm{Cut}(S,T)}{\mathrm{Vol(S)}} + \frac{\mathrm{Cut(S,T)}}{\mathrm{Vol(T)}}, \tag{5.1}$$

where $\mathrm{Cut}(S,T)$ is the number of edges with one node in T and the other in S, and $\mathrm{Vol}(S)$ (resp. for T) is the number of edges with at least one end in S (resp. T). We first calculate the volume of S, T. By Figure 3 we see that S have 3 edges with both ends in S and 2 edges with one end in S. Therefore, $\mathrm{Vol}(S) = 3 + 2 = 5$. By symmetry it follows that $\mathrm{Vol}(T) = 5$. To calculate $\mathrm{Cut}(S,T)$ we see that there are only two edges with one end in S and the other in T. Hence, $\mathrm{Cut}(S,T) = 2$. Substituting in (5.1) we conclude that

$$\mathrm{NC}(S,T) = \frac{2}{5} + \frac{2}{5} = \frac{4}{5}.$$

5.2 Modularity

Compute the modularity of the following partitions:

(i) $S = \{1, 2, 3\}$, $T = \{4, 5, 6\}$,

(ii) $A_1 = \{1\}$, $A_2 = \{2, 3\}$, $A_3 = \{4, 5\}$, $A_4 = \{6\}$,

for the graph in 3.

Solution: (i) Before we compute the modularity for the partition $S = \{1,2,3\}$, $T = \{4,5,6\}$ we first state how it is defined. For a graph $G = (V,E)$ denote by k_v the degree of the vertex $v \in V$. Furthermore, denote by A the adjacency matrix of G and $m = |V|$. Suppose now, that the set V partitions into c distinct communities $C_1,\dots,C_c$. Then, the modularity of the partition is given by

$$\begin{aligned} Q &= \frac{1}{2m} \sum_{(u,v)\in V\times V} \left[A_{u,v} - \frac{k_u k_v}{2m} \right] \cdot \delta(c_u, c_v) \\ &= \sum_{i=1}^{c} (e_{ii} - \alpha_i^2), \end{aligned} \tag{5.2}$$

where e_{ii} is the fraction of edges with one end vertices in community i and other in community j; i.e.

$$e_{ij} = \sum_{(u,v)\in V\times V} \frac{A_{uv}}{2m} \cdot \mathbb{1}_{u\in C_i} \cdot \mathbb{1}_{v\in C_j}, \tag{5.3}$$

where by $\not\Vdash_{u\in C_i}$ we mean that $\mathbb{1}_{u\in C_i} = 1$ iff $u \in C_i$ otherwise $\mathbb{1}_{u\in C_i} = 0$. And finally,

$$\alpha_i = \frac{k_i}{2m} = \sum_{j=1}^{|V|} e_{ij}. \tag{5.4}$$

Considering the equations (5.2), (5.3), (5.4) we calculate for the partition $S = \{1,2,3\}, T = \{4,5,6\}$ that

$$\begin{aligned} e_{SS} &= \sum_{(u,v)\in V\times V} \frac{A_{uv}}{2m} \cdot \mathbb{1}_{u\in S} \cdot \mathbb{1}_{v\in S} \\ &= 2 \cdot \frac{3}{2m} = \frac{3}{m} = 3/8. \end{aligned}$$

By symmetry, it follows that $e_{TT} = 3/8$. Now, there are only two edges with one end in S and the other in T, therefore again by symmetric $e_{ST} = e_{TS} = 1/2m + 1/2m = 1/8$. Therefore, by (5.2) we conclude that

$$\begin{aligned} Q &= (e_{SS} - \alpha_S^2) + (e_{TT} - \alpha_T^2) \\ &= 2e_{SS} - 2\alpha_S^2 = 2e_{SS} - 2(e_{SS} + e_{ST})^2 \\ &= \frac{3}{4} - 2\frac{1}{4} = \frac{3}{4} - \frac{1}{2} = \frac{1}{4}. \end{aligned}$$

(ii) Now, for the partition $A_1 = \{1\}, A_2 = \{2,3\}, A_3 = \{4,5\}, A_4 = \{6\}$, by symmetry we have that $e_{11} = e_{44} = 0$ and $e_{22} = e_{33} == 1/8$. Moreover, again by symmetry $\alpha_1 = \alpha_4$ and $\alpha_2 = \alpha 3$. Now,

$$\begin{aligned} \alpha_1 &= e_{11} + e_{12} + e_{13} + e_{14} = 1/8 + 1/16 = 3/16, \\ \alpha_2 &= e_{21} + e_{22} + e_{23} + e_{24} = 1/8 + 1/8 + 1/16 = 5/16. \end{aligned}$$

Therefore, the modularity of the partition is equal to

$$
\begin{aligned}
Q &= (e_{11} - \alpha_1^2) + (e_{22} - \alpha_2^2) + (e_{33} - \alpha_3^2) + (e_{44} - \alpha_4^2) \\
&= -\alpha_1^2 - \alpha_4^2 + (e_{22} - \alpha_2^2) + (e_{33} - \alpha_3^2) \\
&= -2\alpha_1^2 + 2(e_{22} - \alpha_2^2) \\
&= 2\frac{9}{16^2} + \frac{2}{8} - 2\frac{25}{16^2} = \frac{-18 - 50 + 64}{16^2} \\
&= -\frac{4}{16^2} = -\frac{1}{64}.
\end{aligned}
$$

6 Advertising on Web

6.1 Exercise 8.2.1 in MMDS

Design an on-line algorithm for the ski-buying problem that has the best possible competitive ratio. What is that competitive ratio?

Solution: Before we state the on-line algorithm with the optimal competitive ratio, we first introduce some notation. By A we denote an on-line algorithm for the ski-buying problem and by $\mathscr{A}$ is the family of all possible on-line algorithms for the *ski-buying* problem. The optimal off-line algorithm is denoted by M. By $\text{cost}(A)$ we denote the cost of the on-line algorithm $A \in \mathscr{A}$. In the *ski-buying* problem for each day $t = 1, 2 \ldots$, we have only two options: either to go for ski or not to. We denote by 1 the action of going to ski and by 0 otherwise. Since, by the nature of the problem, if an action 0 occurs at a given day t we will never go for skiing, we can denote a series of n actions by $(1, \ldots, 1, 0, 0, 0, \ldots)$. With this notation we can think of the $\text{cost}(A)$ of an on-line algorithm $A \in \mathscr{A}$ as function $\text{cost}_A : S \to \mathbb{R}_+$, where

$$S = \left(\bigcup_{i=1}^{\infty} \left(\{1\}^{[i]} \times \{0\}^{\mathbb{N} \setminus [i]} \right) \right) \bigcup \{1\}^{\mathbb{N}},$$

$[i] = \{1, \ldots, i\}$.[3] We now define the competitive ratio of an on-line algorithm $A \in \mathscr{A}$.

Definition 1 (Competitive ratio)**.** Let $A \in \mathscr{A}$ be an on-line algorithm for the *ski-buying* problem and M be the optimal off-line algorithm. The competitive ratio c_A of the algorithm A with respect to M is given by the ratio

$$c_A = \sup_{s \in S} \frac{\text{cost}_A(s)}{\text{cost}_M(s)}. \tag{6.1}$$

Our goal is to determine an on-line algorithm $A \in \mathscr{A}$ with smallest possible competitive ratio c_A. In other words, our goal is to find $A^* = \underset{A \in \mathscr{A}}{\arg\min}\, c_A$. Before we proceed by determining A^* let us make a few remarks about the ratio in (6.1). First, observe that since for every $s \in S$ we have that $\text{cost}_A(s) \geq \text{cost}_M(s)$ it follows that $c_A \geq 1$ for every $A \in \mathscr{A}$. Now, there are cases where c_A might be $+\infty$. Indeed, suppose that A is the on-line algorithm for which we decide to rent every day we decide to go for skiing. Then, for the series of actions

$$s_n = (\underbrace{1, \ldots, 1}_{n \text{ times}}, 0)$$

we have that $\text{cost}_A(s_n) = 10 \cdot n$. But for $n \geq 10$ the optimal algorithm M has cost equal to $\text{cost}_M(s_n) = 100$. Therefore, for every $n \geq 10$ we have that

$$c_A \geq \frac{\text{cost}_A(s_n)}{\text{cost}_M(s_n)} = \frac{n}{10},$$

[3] The set $\{1\}^{\mathbb{N}}$ corresponds to the series of actions that we choose to go for ski every day.

letting $n \to \infty$ we conclude that $c_A = +\infty$. Now, we claim that the algorithm A^* with the minimum competitive ratio c_A is the one that rents a ski for the first $1 \leq i \leq 9$ days we decide to go skiing and buys the ski for 100$ if we decide go skiing for more than 9 days. We claim that $c_{A^*} = 1.9$. To this end, we will calculate the cost of A^* in each element of S. For $s = (\underbrace{1,\dots,1}_{n \leq 9},0)$ we have that $\text{cost}_A(s) = 10 \cdot n$. Since, $\text{cost}_M(s) = 10 \cdot n$ we have that $\text{cost}_A(s) = \text{cost}_M(s)$. Now, for $s_n = (\underbrace{1,1,\dots,1}_{n},0,0,\dots)$, $n \geq 10$ we have that $\text{cost}_{A^*}(s_n) = 190$ and $\text{cost}_M(s_n) = 100$. Therefore,

$$c_{A^*} \geq \frac{\text{cost}_{A^*}(s_n)}{\text{cost}_M(s_n)} = 1.9.$$

Finally, the last element $s \in S$ is $s = (1,1,\dots,1,\dots)$ for which we have that $\text{cost}_{A^*}(s) = 190$ and $\text{cost}_M(s) = 100$. Hence, we conclude that $c_{A^*} = 1.9$. Now, to prove that A^* has the minimum competitive ratio we consider an arbitrary on-line algorithm $A \in \mathscr{A}$ and we prove that $c_A \geq c_{A^*}$. Since we showed that the algorithm that corresponds to the strategy of renting a ski every day has competitive ratio equal to $+\infty$ we can w.l.g assume that A buys the ski for 100$ at some point. Suppose that this happens at day $1 \leq i^* < \infty$. The case where $i^* = 10$ corresponds to the algorithm A^*. Now, fix $1 \leq i^* \leq 9$ and let $s_{i^*} = (\underbrace{1,\dots,1}_{i^*},0,0,\dots)$. We have that $\text{cost}_A(s_{i^*}) = 10 \cdot (i^* - 1) + 100$ and $\text{cost}_M(s_{i^*}) = 10 \cdot i^*$. Therefore,

$$c_A \geq \frac{\text{cost}_A(s)}{\text{cost}_M(s)} = \frac{10 \cdot (i^*-1) + 100}{10 \cdot i^*} = \frac{i^*-1}{i^*} + \frac{10}{i^*}. \tag{6.2}$$

Let $J(i^*) := (i^*-1)/i^* + 10/i^*$, for $i^* = 1,\dots,9$. Then, we have the following table.

i^*	1	2	3	4	5	6	7	8	9
$j(i^*)$	10	5.5	4	3.25	2.8	2.5	2.28	2.125	2

By examining all possible values we conclude that $J(i^*) > 1.9$ for all $i^* = 1,\dots,9$. Which means that the competitive ratio c_A of A is strictly larger than $c_{A^*} = 1.9$ for $i^* = 1,\dots,9$. Suppose now, that $i^* > 10$. Then, again for the sequence s_{i^*} we have that

$$\begin{aligned} c_A \geq \frac{\text{cost}_A(s)}{\text{cost}_M(s)} &= \frac{10 \cdot (i^*-1) + 100}{100} \\ &= \frac{i^*-1}{10} + 1 > 0.9 + 1 = 1.9, \end{aligned}$$

since $i^* > 10 \implies i^* - 1 > 9$. Therefore, we deduce that A^* is indeed the optimal on-line algorithm for the *ski-buying* problem with respect to the competitive ratio.

7 Recommendation Systems

7.1 Exercise 9.2.3 in MMDS

The computers' features presented in exercise 9.2.2 in MMDS are given in the following table.

Table 1: Feature vectors for three computers A, B and C.

Feature	A	B	C
Processor speed	3.06	2.68	2.92
Disk size	500	320	640
Main-Memory size	6	4	6

(a) The rating for the user are: $A : 4$ stars, $B : 2$ stars and $C : 5$ stars. Therefore, the mean values according to these ratings is

$$\mu = \frac{4+2+5}{3} = \frac{11}{3}.$$

Therefore, after normalizing the ratings for this user we obtain that the feature vector becomes

$$(A-\mu, B-\mu, C-\mu) = (1/3, -5/3, 4/3).$$

(b) The user profile $P(f)$ for each feature f is given by

$$P(f) = \sum_{i=A,B,C} f(i) \cdot \text{Norm}(i),$$

therefore we have

$$\begin{aligned} P(\text{Processor speed}) &= 3.06 \cdot \frac{1}{3} - 2.68 \cdot \frac{5}{3} + 2.92 \cdot \frac{4}{3} = \frac{67}{150} \\ P(\text{Disk size}) &= 500 \cdot \frac{1}{3} - 320 \cdot \frac{5}{3} + 640 \cdot \frac{4}{3} = \frac{1460}{3} \\ P(\text{Main-Memory size}) &= 6 \cdot \frac{1}{3} - 4 \cdot \frac{5}{3} + 6 \cdot \frac{4}{3} = \frac{10}{3}. \end{aligned}$$

7.2 Exercise in 9.3.2 in MMDS

Below is the utility matrix given in MMDS.

Table 2: The utility matrix.

	a	b	c	d	e	f	g	h
A	4	5		5	1		3	2
B		3	4	3	1	2	1	
C	2		1	3		4	5	3

(a) By replacing all 3's, 4's and 5's by 1 and replace 1's, 2's, and blanks by 0 we have that Table 2 becomes

Table 3: The transformed utility matrix.

	a	b	c	d	e	f	g	h
A	1	1	0	1	0	0	1	0
B	0	1	1	1	0	0	0	0
C	0	0	1	1	0	1	1	1

Then, all pairwise Jaccard distances are evaluated as

$$d_J(i,j) = 1 - J(i,j) = 1 - \frac{|i \cup j|}{|i \cap j|}.$$

The results are shown in the table below

Table 4: The transformed utility matrix.

	a	b	c	d	e	f	g	h
a	0							
b	1/2	0						
c	1	1/2	0					
d	2/3	1/3	2/3	0				
e	1	1	1	1	0			
f	1	1	1	2/3	1	0		
g	1/2	2/3	1	1/3	1	1/2	0	
h	1	1	1	2/3	1	0	1/2	0

To perform the hierarchical clustering we start by forming all singletons and then we combine each cluster by using the pairwise distances from Table 4. The clusters consisting of singletons are the following:

$$\mathscr{C}_1 = \{a\}, \quad \mathscr{C}_2 = \{b\}, \quad \mathscr{C}_3 = \{c\}, \quad \mathscr{C}_4 = \{d\}, \quad \mathscr{C}_5 = \{e\}, \quad \mathscr{C}_6 = \{f\}, \quad \mathscr{C}_7 = \{g\}, \quad \mathscr{C}_8 = \{h\}.$$

Then, the minimum distance occurs between $\{f\},\{h\}$ which is zero. Therefore, in the next step the clusters are

$$\mathscr{C}_1 = \{a\}, \quad \mathscr{C}_2 = \{b\}, \quad \mathscr{C}_3 = \{c\}, \quad \mathscr{C}_4 = \{d\}, \quad \mathscr{C}_5 = \{e\}, \quad \mathscr{C}_6 = \{f,h\}, \quad \mathscr{C}_7 = \{g\}.$$

The next minimum distance occurs between b,d which is $1/3$. Therefore, $\mathscr{C}_1,\mathscr{C}_2$ are merged into two clusters giving us

$$\mathscr{C}_1 = \{a\}, \quad \mathscr{C}_2 = \{b,d,g\}, \quad \mathscr{C}_3 = \{c\}, \quad \mathscr{C}_4 = \{e\}, \quad \mathscr{C}_5 = \{f,h\}.$$

The next minimum distance is $1/2$ and there are several options for merging the clusters with this distance. We choose the clusters $\mathscr{C}_1,\mathscr{C}_2$ to be merged, leading to

$$\mathscr{C}_1 = \{a,b,d,g\}, \quad \mathscr{C}_2 = \{c\}, \quad \mathscr{C}_3 = \{e\}, \quad \mathscr{C}_4 = \{f,h\}.$$

(b) In order to compute the average cluster score per user, we only average over the non-blank entries for each user. For example, the average score $\mathscr{C}_1$ for user A will be $(4+5+5+3)/4 = 17/4$. Performing the same calculation for each user and each of the clusters $\mathscr{C}_1,\mathscr{C}_2,\mathscr{C}_3,\mathscr{C}_4$ we obtain the following table.

Table 5: User scores per cluster.

	$\mathscr{C}_1$	$\mathscr{C}_2$	$\mathscr{C}_3$	$\mathscr{C}_4$
A	17/4	0	1	2
B	7/3	4	1	2
C	10/3	1	0	7/2

(c) Given the results of Table 5, the vector norms for each user are

$$\|A\| = \frac{\sqrt{369}}{4}, \quad \|B\| = \frac{\sqrt{238}}{3}, \quad \|C\| = \frac{\sqrt{877}}{6}.$$

Based on the above norms we can now calculate the cosine distance between two users as follows:

$$1-\cos(A,B) = 1 - \frac{\frac{17}{4}\cdot\frac{7}{3}+1+4}{\frac{\sqrt{364}}{4}\cdot\frac{\sqrt{238}}{3}} = 1 - \frac{179}{\sqrt{87822}} \approx 1-0.604 = 0.396.$$
$$1-\cos(A,C) = 1 - \frac{\frac{17}{4}\cdot\frac{10}{3}+2\cdot\frac{7}{2}}{\frac{\sqrt{369}}{4}\cdot\frac{\sqrt{877}}{6}} = 1 - \frac{508}{\sqrt{323613}} \approx 1-0.893 = 0.107$$
$$1-\cos(B,C) = 1 - \frac{\frac{7}{3}\cdot\frac{10}{3}+4\cdot 1+2\frac{7}{2}}{\frac{\sqrt{238}}{3}\cdot\frac{\sqrt{877}}{6}} = 1 - \frac{338}{\sqrt{208726}} \approx 1-0.740 = 0.260.$$

References

[1] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2022.

[2] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[3] Julian Vu and Katerina Potika. “Edge Betweenness Centrality on Trees”. In: *2020 Second International Conference on Transdisciplinary AI (TransAI)*. IEEE. 2020, pp. 104–107.