Algorithmic Data Science - Exercise Series 2

Konstantinos Papadakis
Data Science and Machine Learning 03400149
konstantinospapadakis@mail.ntua.gr

June 25, 2022

Exercise 1

(a)

We have that

$$\begin{aligned} & h_{a,b}(x) = h_{a,b}(y) \\ \iff & ax + b \equiv ay + b \mod m \\ \iff & a(x - y) \equiv 0 \mod m \end{aligned}$$

which in the case of $x = m,\ y = 0$ is true $\forall a, b$ therefore

$$P(h_{a,b}(x) = h_{a,b}(y)) = 1 > \frac{1}{m}$$

meaning that the family is not universal.

(b)

This exercise is *Theorem 11.5* in [1].

Let $x, y \in \mathbb{Z}_p : x \neq y$.

Define

$$\begin{aligned} u &:= ax + b \mod p \\ v &:= ay + b \mod p \end{aligned}$$

We have that $u \neq v$ since $u - v \equiv a(x - y) \not\equiv 0 \mod p$ because $a \neq 0$, $x - y \not\equiv 0 \mod p$, and $\mathbb{Z}_p$ is a field. Note that $x - y \not\equiv 0 \mod p$ holds because by hypothesis $x \neq y$ and $x, y < p$.

Therefore, there are no collisions when we apply $x \mapsto ax + b \mod p$.

We proceed to show that $(a, b) \mapsto (ax+b \mod p,\ ax+b \mod p)$ is a bijection between the pairs $(a, b) \in \mathbb{Z}_p^* \times \mathbb{Z}_p$ and the pairs $(u, v) \in \mathbb{Z}_p \times \mathbb{Z}_p : u \neq v$.

We can solve for a, b and get a unique solution

$$\begin{aligned} a &= \frac{u - v}{x - y} \mod p \\ b &= r - ak \mod p \end{aligned}$$

Where $\frac{1}{t}$ is the inverse of t in $\mathbb{Z}_p$

Therefore the mapping is one to one. Since we also have that both the domain and the co-domain have $p(p - 1)$ elements, the mapping is a bijection. Thus, if (a, b) is uniformly distributed, so is (u, v).

Therefore, the probability that $x, y \in \mathbb{Z}_p : x \neq y$ collide is equal to the probability that $u \equiv v \mod m$ collide when $(u, v) \in \mathbb{Z}_p \times \mathbb{Z}_p : u \neq v$ are chosen uniformly randomly. We proceed to calculate that probability.

Given u, of the $p - 1$ possible remaining values for v we have that at most $\lceil \frac{p}{m} \rceil - 1 \leq \frac{p-1}{m}$ can collide with u.

Therefore the probability of collision is $\leq \frac{1}{m}$, meaning that the hash function family is universal.

(c)

The proof in is still valid, since $x \in U \implies x < p$ is still valid. Therefore the hash function family remains universal.

Exercise 2

(a)

From what I understand, the expected number of probes for a successful search is not equal to the expected number of probes for an insertion. What [1] says in the proof of *Corollary 11.7* is that the expected number of probes for an *Unsuccessful* search is equal to the expected number of probes for an insertion, and that number is bounded above by $\frac{1}{1-a}$ where a is the load factor.

The expected number of probes a for a successful search on the other hand is bounded above by $\frac{1}{a} \ln \frac{1}{1-a}$, as shown in *Theorem 11.8*.

If the expected values were equal, wouldn't the book bound them by the same number?

(b)

First we need to show that the expected number of probes in an unsuccessful search is bounded by $\frac{1}{1-a}$.

Let X be a random variable describing the number of probes in an unsuccessful search.

We have

$$\begin{aligned} \mathrm{E}[X] &= \sum_{i=0}^{\infty} i \Pr(X = i) \\ &= \sum_{i=1}^{\infty} \Pr(X \geq i) \end{aligned}$$

Now,

$$\begin{aligned} \Pr(X \geq i) &= \frac{n}{m} \cdot \frac{n-1}{m-1} \cdots \frac{n-i+2}{m-i+2} \\ &\leq \left(\frac{n}{m}\right)^{i-1} \\ &= a^{i-1} \end{aligned}$$

Therefore,

$$\mathrm{E}[X] \leq \sum_{i=1}^{\infty} a^{i-1} = \frac{1}{1-a}$$

Since an element is inserted only if there is room in the table (thus $a < 1$), inserting a key requires an unsuccessful search followed by placing the key into

the first empty slot found. Thus, the expected number of probes for an insertion is at most $\frac{1}{1-a}$.

We now proceed to prove the result about the successful search.

A search for a key reproduces the same probe sequence as when the element with key was inserted. By the above result, if the element was the $(i+1)$st element inserted, then the expected number of probes made in a search for it is at most $\frac{1}{1-1/m} = \frac{m}{m-i}$.

Therefore the expected number of probes for a successful search is

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

Exercise 3

(a)

We proceed to calculate the time complexity of the Girvan-Newman algorithm, shown in Algorithm 1. We iterate over $|V|$ nodes, so everything will be multiplied by $|V|$. The BFS step takes $O(|V|+|E|)$ time. The accumulation with DFS also takes $O(|V|+|E|)$ time. The betweenness calculation takes $O(|V|+2|E|) = O(|V|+|E|)$ time as well. Therefore, we end up with $O(|V|(|V|+|E|))$ time.

(b)

In the case of a tree, since a tree is by definition an undirected acyclic graph, it can also be viewed as a DAG, meaning that we don't need to perform the BFS step. Also, in the second DFS part, since tree nodes have only one parent, we don't need to compute the `u.npaths/s` factor because it's always equal to 1. The complexity doesn't change in the end, since the DFS still takes $O(|V|+|E|)$ time. Finally in a tree we have that $|E| = |V| - 1$. Thus the complexity can now be written as $O(|V|^2)$.

Algorithm 1 Girvan-Newman

```
for each e in E do
    e.betweenness ← 0
end for

for each r in V do
    G ← DAG by performing BFS from r
    for each v in G do
        v.npaths ← 1
    end for
     Cumulatively sum node npaths from leaves to root (post-order DFS)
    for each v in G do
        v.credit ← 1
    end for
    for v in post-order DFS(r) do
        s ← 0
        for neighbor u of v incoming from above do
            s ← s + u.npaths
        end for
        for each edge e incoming to v from the above node u do
            e.tempbetweenness ← v.credit * u.npaths / s
            u.credit ← u.credit + e.tempbetweenness
        end for
    end for
    for each e in E do
        e.betweenness ← e.betweenness + e.tempbetweenness/2
    end for
end for
```

Exercise 4

Exercise 10.4.1 of [2]

The adjacency matrix is

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

The degree matrix is

$$D = \mathrm{diag}(2, 3, 3, 3, 3, 2, 3, 3, 2)$$

The Laplacian matrix is

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

Exercise 10.4.2 of [2]

Using Wolfram Mathematica's function `Eigensystem` we find out that second smallest eigenvalue of L after 0 is $\frac{5-\sqrt{13}}{2}$ with corresponding eigenvectors

$$v_1 = \begin{pmatrix} \frac{5+\sqrt{13}}{2} \\ \frac{3+\sqrt{13}}{2} \\ \frac{1+\sqrt{13}}{2} \\ -\frac{7+\sqrt{13}}{1+\sqrt{13}} \\ \frac{-3-\sqrt{13}}{2} \\ -\frac{3\left(3+\sqrt{13}\right)}{1+\sqrt{13}} \\ -1 \\ 1 \\ 0 \end{pmatrix}, \qquad v_2 = \begin{pmatrix} -\frac{2\left(4+\sqrt{13}\right)}{1+\sqrt{13}} \\ \frac{-1-\sqrt{13}}{2} \\ -2 \\ 1 \\ 2 \\ \frac{7+\sqrt{13}}{1+\sqrt{13}} \\ \frac{\sqrt{13}-1}{2} \\ 0 \\ 1 \end{pmatrix}$$

The partition is based on the signs of the coordinates of the eigenvector. Eigenvector v_1 partitions the graph into $\{A, B, C, H, I\}$ and $\{D, E, F, G\}$, while eigenvector v_2 partitions the graph into $\{A, B, C\}$ and $\{D, E, F, G, H, I\}$.

Exercise 5

(a)

We have $S = \{1, 2, 3\},\ T = \{4, 5, 6\}$, so the normalized cut value is

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

(b)

The modularity is

$$Q := \frac{1}{2m} \sum_{v,w} \left(A_{vw} - \frac{k_v k_w}{2m} \right) \delta(v, w)$$

where k_v is the degree of vertex v, A is the adjacency matrix, $\delta(u, v)$ is 1 if u and v belong to the same community else 0, and m the number of edges.

Our k's are $3, 2, 3, 3, 2, 3$. Due to the symmetry of the partition we only need to compute the first half and then multiply by two.

$$\begin{aligned} Q &= 2\frac{1}{2m} \left(\sum_{\substack{i,j \in \{1,2,3\} \\ i \neq j}} \left(1 - \frac{k_i k_j}{2m} \right) - \sum_{i \in \{1,2,3\}} k_i^2 \right) \\ &= \frac{1}{8} \left(2 \left((1 - \frac{3 \cdot 2}{16}) + (1 - \frac{3 \cdot 3}{16}) + (1 - \frac{2 \cdot 3}{16}) \right) - \frac{1}{16} \left(3^2 + 2^2 + 3^2 \right) \right) \\ &= \frac{1}{4} \end{aligned}$$

Considering the cut $\{1\}, \{2, 3\}, \{4, 5\}, \{6\}$, we have

$$\begin{aligned} Q &= 2\frac{1}{16} \left(-\frac{k_1^2 + k_2^2 + k_3^2}{16} + 2 \left(1 - \frac{k_2 k_3}{16} \right) \right) \\ &= \frac{1}{8} \left(-\frac{3^2 + 2^2 + 3^2}{16} + 2 \left(1 - \frac{2 \cdot 3}{16} \right) \right) \\ &= -\frac{1}{64} \end{aligned}$$

Exercise 6

Exercise 8.2.1 of [2]

First of all, we assume that the prices for renting and purchasing are not variable, or at least that the rent price divides the purchase price.

Consider the strategy A where we rent until we realize we should have bought, then buy. (In our case this means we rent 9 times, then buy). In symbols, if p is the purchase price and r is the rent price, we rent $\lceil \frac{p}{r} \rceil$ times, and then we buy.

We analyze the cost of A, when r divides p.

- In the case (subset of all inputs) where we ski fewer than p/r times, then our strategy is optimal.

- In the case where we ski p/r or more times, then the optimal strategy would be to buy from the beginning, paying only p. Strategy A on the other hand, would pay $(p/r - 1)r + p = p - r + p = 2p - r$.

So the optimal strategy costs us p while strategy A, in the worst case, costs us $2p - r$.

Since we are talking about costs instead of earnings, it's natural to use the competitive *cost* ratio (the lower the better). Strategy A has competitive cost ratio $\frac{2p-r}{p} = 2 - \frac{r}{p}$. In our case this is $2 - \frac{10}{100} = 1.9$.

We now proceed to prove that if r divides p, then the strategy A is optimal among all other deterministic online strategies.

Consider an alternate deterministic strategy B.

- If in B we never purchase, then the cost is infinite.
- Assume that B purchases at some point.
  - If the purchase always happens before or at $p/r - 1$, say day $k \leq p/r - 1$, then in the worst case we pay p instead of krs, which is worse than the worst case for A.
  - If on the other hand the purchase always happens after p/r, say on day $k > p/r$, then in the worst case we pay $(k - 1)r + p > (p/r - 1)r + p = 2p - r$ instead of p, which is worse than the worst case for A.

Therefore strategy A is optimal among all other deterministic online strategies, if r divides p.

Exercise 8.4.1 of [2]

Whatever we do C, will be assigned 2 queries. The worst case is to assign x to B (or C), then assign x to C (or B), leaving A with no assignments; then we are left with two slots which cannot be filled The other worst cases work similarly, that is we assign a specific type of query to advertisers that bid on multiple types of queries, instead of assigning them to advertisers that bid only on that specific query.

The greedy algorithm can assign the query yyzz to C, C, None, None, while the optimal algorithm assigns to B, B, C, C. The greedy algorithm in this case assigned only half the number of queries compared to the number of queries that the optimal algorithm assigned.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

[2] J. Leskovec, A. Rajaraman, and J. Ullman, *Mining of Massive Datasets*, 3rd ed. Cambridge University Press, 2019.