University of Edinburgh

INFR11156: Algorithmic Foundations of Data Science (2025)

Guide to Examinable Materials

This guide organised by topics is to give you an idea about the depth of understanding of the materials required for the final exam. The examinable material is classified as follows:

- Definition: This is an important definition of a concept or a problem – you should certainly know it.
- Statement: You need to know the statement of the result. We may ask you what the result is and/or you may need to know it to answer one of the questions. In the case of a Lemma or Theorem, you won't be asked for the proof. For an Algorithm you should know which problem it solves and its performance (time and space complexity etc.). You won't be asked how it works or to prove its run-time or correctness.
- Method: You need to know everything from Statement and also what the method is. That is we may ask you how it works but we won't ask you to prove it.
- Proof: You need to know everything about this. We might ask you what it is, how it works etc. However, the AFDS final exam is not a memory test and, instead of memorising the proofs, you should understand why the statements are proven in a certain way: we may ask you whether a different proof technique works or not.

During the exam, your aim is to convince us that you know why something is true. Be as rigorous and clear as you can. Remember we will be marking as fairly as we can — if we believe you understand why something is true you will get good marks.

Tips:

- Calculators may be used in this examination.
- Question 1 is relatively easier than Question 2 and 3. You should answer Q1 first, but leave enough time for Q2 or Q3.
- If both Q2 and Q3 are answered, then only Q2 will be marked. Given the difficulty of Q2 and Q3, you should spend time checking both questions carefully, and decide which question to answer.
- A sample format of the questions in the final exam:
  - Describe the algorithm we discussed in class.
  - Present the outcome of some algorithm, e.g., given an undirected graph and input vector, calculate the vertex set that the sweep-set based algorithm outputs.
  - Prove or disprove by counterexample some statement like "when doubling the weights of all the edges in a graph G, the largest eigenvalue of the normalised Laplacian matrix of the new graph is doubled."

1 High-Dimensional Spaces

The law of large numbers. Most of our analysis is based on the following three basic probability inequalities. Let X be a non-negative random variable. Then the Markov inequality states that, for any $c > 0$, we have

$$\mathbf{Pr}\left[X \geq c\right] \leq \frac{\mathbf{E}\left[X\right]}{c}. \qquad \text{[Statement]}$$

The Chebyshev's Inequality states that, for any $c > 0$, we have

$$\mathbf{Pr}\left[\left|X - \mathbf{E}\left[X\right]\right| > c\right] \leq \frac{\mathbf{Var}\left[X\right]}{c^2}. \qquad \text{[Statement]}$$

Based on the inequality above, we can prove the so-called Law of Large Numbers, i.e., for any n independent samples of a random variable X denoted by $x_1, \cdots x_n$, we have

$$\mathbf{Pr}\left[\left|\frac{x_1 + \cdots + x_n}{n} - \mathbf{E}\left[X\right]\right| \geq \varepsilon\right] \leq \frac{\mathbf{Var}\left[X\right]}{n\varepsilon^2}. \qquad (1) \quad \text{[Statement]}$$

Notice that n is in the denominator of the RHS in (1), which implies that the more samples we take, the smaller the error we have; also notice that ε is in the denominator of the RHS in (1), which implies that the large the value of ε, the smaller the error is.

Application of the law of large numbers. To see the application of the law of large numbers, we assume that $y = (y_1, \cdots, y_d)$ and $z = (z_1, \cdots, z_d)$ are two random points drawn from $d-$dimensional random Gaussian with unit variance in each direction. Then, it holds that

$$\mathbf{E}\left[y_i^2\right] = \mathbf{E}\left[\left|y_i - \mathbf{E}\left[y_i\right]\right|^2\right] = \mathbf{Var}[y_i] = 1 \qquad \text{[Proof]}$$

and $\mathbf{E}\left[z_i^2\right] = 1$ for the same reason. By linearity of expectation, we have $\mathbf{E}\left[\|y\|^2\right] = d$ and $\mathbf{E}\left[\|z\|^2\right] = d$. By the Law of Large Numbers we know that $\|y\|^2 \approx d$ and $\|z\|^2 \approx d$ with high probability. On the other hand, we have that

$$\mathbf{E}\left[(y_i - z_i)^2\right] = \mathbf{E}\left[y_i^2 - 2 \cdot y_i \cdot z_i + z_i^2\right] = \mathbf{E}\left[y_i^2\right] - 2 \cdot \mathbf{E}\left[y_i\right] \cdot \mathbf{E}\left[z_i\right] + \mathbf{E}\left[z_i^2\right] = 2.$$

This gives us that

$$\|y - z\|^2 \approx 2d \approx \|y\|^2 + \|z\|^2,$$

i.e., y and z must be approximately orthogonal. [Method]

Dimension reduction. The Johnson-Lindenstrauss lemma states that any n points in high-dimensional Euclidean space can be mapped onto k dimensions where $k = O\left(\log n/\varepsilon^2\right)$ without distorting the Euclidean distance between any pair of points more than a factor of $1 \pm \varepsilon$. Formally, for any $X \subseteq \mathbb{R}^d$ of n points and $\varepsilon \in (0, 1/5)$, there is a random matrix $\Phi \in \mathbb{R}^{k \times d}$, such that it holds with constant probability that

$$\forall x, y \in X: \qquad (1 - \varepsilon)\|x - y\|_2 \leq \|\Phi x - \Phi y\|_2 \leq (1 + \varepsilon)\|x - y\|_2,$$

where $k = O\left(\log n/\varepsilon^2\right)$. [Proof]

2 Best-Fit Subspaces and Singular Value Decomposition

Singular values and singular vectors. Given matrix $A \in \mathbb{R}^{n\times d}$, the first singular vector v_1 of A is defined as

$$v_1 \triangleq \arg\max_{\|v\|=1} \|Av\|,$$

and the value $\sigma_1(A) \triangleq \|Av_1\|$ is called the first singular value of A. The second singular vector v_2 is defined by

$$v_2 \triangleq \arg\max_{v\perp v_1, \|v\|=1} \|Av\|,$$

and $\sigma_2(A) \triangleq \|Av_2\|$ is called the second singular value of A. The other singular values and singular vectors can be defined inductively in the same way. [Definition]

To find the best-fit subspace, we present a simple greedy algorithm. Namely, the greedy algorithm finds v_1 that maximises $\|Av\|$ and then the best-fit 2-dimensional subspace containing v_1, etc. We also prove the correctness of this greedy strategy, i.e., for any $A \in \mathbb{R}^{n\times d}$ with singular vectors $v_1, \ldots, v_r$, and $1 \le k \le r$, then the subspace spanned by $v_1, \ldots, v_k$ is the best-fit k-dimensional subspace for A. [Proof]

Singular value decomposition. We can think of $A \in \mathbb{R}^{n\times d}$ as a linear transformation taking a vector v_1 in its row space to a vector $u_1 = Av_1$ in its column space. Many applications require to find an orthogonal basis for the row space and transform it into an orthogonal basis for the column space: $Av_i = \sigma_i u_i$. The heart of the problem is to find $v_1, \ldots, v_r$ for the row space of A for which

$$\begin{aligned} A[v_1, v_2, \ldots, v_r] &= [\sigma_1 u_1, \sigma_2 u_2, \ldots, \sigma_r u_r] \\ &= [u_1, u_2, \ldots, u_r] \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{pmatrix}. \end{aligned} \tag{2}$$

Then, it is easy to see that the left and right-singular vectors $u_i = \frac{1}{\sigma_i} Av_i$, v_i, and their associated singular values σ_i satisfy (2). With these vectors u_is, v_is, and the singular values σ_is, we can write A in matrix notation as

$$A = UDV^\intercal, \qquad \text{[Statement]}$$

where u_i is the i-th column of U, $v_i^\intercal$ is the i-th row of $V^\intercal$, and D is the diagonal matrix with σ_i as the i-th entry on its diagonal. This factorisation of A in the form of $UDV^\intercal$ is called *Singular value decomposition*. It is easy to check that

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^\intercal. \qquad \text{[Statement]}$$

The power method for computing the largest eigenvalue. The main ideas of the power method for computing eigenvalues and eigenvectors are summarised as follows: instead of computing B^k, we select a random vector x and compute $B^k x$. For time complexity, notice that computing Bx for any vector x takes $O(n + \mathrm{nnz}(B))$ time if the non-zero entries of matrix B are stored by an adjacency list, where $\mathrm{nnz}(B)$ is the number of non-zero entries of matrix B. Hence, the total runtime for computing $B^k x$ is $O(k \cdot (n + \mathrm{nnz}(B)))$. The formal description of the power method for computing λ_1 is shown in Algorithm 1. We also prove that, for any parameter $\varepsilon > 0$, with probability 3/16 over the initial choices of x_0, Algorithm 1 outputs a vector x_k such that [Method]

$$\frac{x_k^\intercal B x_k}{x_k^\intercal x_k} \ge (1-\varepsilon) \cdot \lambda_1 \cdot \frac{1}{1 + 4n(1-\varepsilon)^{2k}}.$$

In particular, when setting $k = O(\log n/\varepsilon)$, we have that

$$\frac{x_k^\intercal B x_k}{x_k^\intercal x_k} \geq (1 - O(\varepsilon))\lambda_1. \qquad \text{[Method]}$$

Algorithm 1 Power method for approximating λ_1

1: **Input:** a PSD symmetric matrix $B \in \mathbb{R}^{n\times n}$, and positive integer k
2: Choose x_0 uniformly at random from $\{-1, 1\}^n$.
3: **for** $i = 1$ to k **do**
4: $\quad x_i = Bx_{i-1}$
5: **end for**
6: **return** x_k

Power method to compute the second largest eigenvalue. Similar with Algorithm 1, we can compute the second largest eigenvalue with the power method as well, by ensuring that the initial vector used for the "power iterations" is perpendicular to v_1, see Algorithm 2 for formal description. For the algorithm's performance, one can prove that with constant probability over the choices of x, Algorithm 2 outputs a vector $y \perp v_1$ such that

$$\frac{y^\intercal M y}{y^\intercal y} \geq \lambda_2 \cdot (1 - \varepsilon) \cdot \frac{1}{1 + 4n(1-\varepsilon)^{2k}},$$

where λ_2 is the second largest eigenvalue of B, counting multiplicities. [Method]

Algorithm 2 Power method for approximating λ_2

1: **Input:** a PSD symmetric matrix $B \in \mathbb{R}^{n\times n}$, and positive integer k
2: Choose x uniformly at random from $\{-1, 1\}^n$.
3: Let $x_0 = x - \langle v_1, x\rangle \cdot v_1$
4: **for** $i = 1$ to k **do**
5: $\quad x_i = Bx_{i-1}$
6: **end for**
7: **return** x_k

Comparison between SVD and JL. We presented a detailed comparison between SVD and JL. You should know which techniques can be applied for specific settings. [Method]

3 Hashing

Most data streaming algorithms rely on constructions of a class of functions, called *hash functions*, that have found a surprising large number of applications. The basic idea behind using hash functions is to make input date have certain independence through some easy-to-compute function. Formally, we want to construct a family of functions $\mathcal{H} = \{h \mid h : U \to M\}$ such that (1) every function $h \in \mathcal{H}$ is easy to represent; (2) for any $x \in U$, $h(x)$ is easy to evaluate; (3) for any set S of small cardinality, hashed values of items in S have small collisions.

We call a set of functions H mapping a universe U to the set $\{0, 1, \ldots, n-1\}$ a universal family of hash functions, if for any pair of elements $u \neq v$, we have

$$\mathbf{P}_{h\in H}\big[h(u) = h(v)\big] \leq 1/n. \qquad \text{[Definition]}$$

To demonstrate the power of universal hash functions, we prove the following result: let H be a universal family of hash functions mapping a universe U to the set $\{0, 1, \ldots, n-1\}$. Let $h \in H$ be chosen uniformly at random from H; let $S \subseteq U$ be a subset of size at most n, and $u \notin S$. Then, the expected number of items in S that collide with u is at most 1. We also present an efficient method for constructing universal hash functions. [Proof] [Proof]

We call a family of functions $H = \{h \mid h : U \mapsto [n]\}$ pairwise independent if, for any h chosen uniformly at random from H, it holds that (1) $h(x)$ is uniformly distributed in $[n]$ for any $x \in U$, and (2) for any $x_1 \neq x_2 \in U$, $h(x_1)$ and $h(x_2)$ are independent. Furthermore, the set $H = \{h : U \to [n]\}$ is call a set of k-wise independent hash functions if for any distinct $x_1, \ldots, x_k \in U$, and any $y_1, \ldots, y_k \in [n]$, [Definition]

$$\mathbf{P}_{h \in H}\big[h(x_1) = y_1 \wedge h(x_2) = y_2 \wedge \cdots \wedge h(x_k) = y_k\big] = \frac{1}{n^k} \quad \text{[Definition]}$$

The family of k-wise independent hash functions can be constructed as follows: let p be a prime, and $k \geq 2$ be an integer. Assume that a seed $s = (a_0, \ldots, a_{k-1})$ is chosen uniformly at random from $\mathbb{Z}_p^k$. Then, the set of functions $H = \big\{h_s | s \in \mathbb{Z}_p^k\big\}$, where

$$h_s(x) = \sum_{i=0}^{k-1} a_i x^i \mod p$$

is k-wise independent. For the case of $k = 2$, we show why this set of functions is pairwise independent. [Method] [Proof]

4 Data Streaming Algorithms

A data stream is a sequence of data $\mathcal{S} = s_1, s_2, \ldots, s_m, \ldots$, where each item s_i belongs to the universe U, where $|U| = n$. A streaming algorithm A takes $\mathcal{S}$ as input and computes some function f of stream $\mathcal{S}$. Moreover, algorithm A has access the input in a "streaming fashion", i.e., A cannot read the input in another order and for most cases A can only read the data once. [Definition]

Depending on how items in U are expressed in S, there are two typical models: (1) Cash Register Model: Each item s_i in stream S is an item of U. Different items come in an arbitrary order. (2) Turnstile Model: In this model we have a multi-set D, and $D = \emptyset$ initially. Every coming item is associated with one of two special symbols in order to indicate the dynamic changes of the data set. For instance, every item in S can be a pair (x, U), and x is added into D if U is "+", and x is deleted from D if U is "−". [Definition]

To cope with the practical applications occurring in processing massive datasets, we need to design *sublinear*-space algorithms that provides a good approximation of certain quantities satisfying the following constraints:

- Space: Space of algorithm A is $O(\text{poly}\log(n))$.
- Quick update time: For every coming item in the stream, quick update time is desired.
- Approximate: For approximation parameter $\varepsilon > 0$ and confidence parameter $\delta > 0$, the output of A achieves a $(1 \pm \varepsilon)$-approximation of the exact value $f(\mathcal{S})$ with probability at least $1 - \delta$. That is, the output $f^\star(\mathcal{S})$ satisfies

$$\mathbf{P}[f^\star(\mathcal{S}) \in [(1-\varepsilon)f(\mathcal{S}), (1+\varepsilon)f(\mathcal{S})]] \geq 1 - \delta. \quad \text{[Statement]}$$

Counting distinct items. Our first problem is to approximate the number of distinct items in a stream. We first define the F_p-norm. Let S be a multi-set, where every item i of S is in $[N]$. Let m_i be the number of occurrences of item i in set S. Then the F_p-norm of set S is defined by

$$F_p \triangleq \sum_{i \in [N]} |m_i|^p,$$

where 0^p is set to be 0. We also introduce the function [Definition]

$$\rho(x) \triangleq \max_i\{i : x \bmod 2^i = 0\},$$

which is be the number of zeros that Binary(x) ends with. Based on the use of $\rho(\cdot)$, the AMS algorithm for approximating F_0 is described in Algorithm 3. We prove that with constant probability the AMS algorithm's output is in $[F_0/3, 3 \cdot F_0]$. [Proof]

Algorithm 3 The ABS algorithm for approximating F_0

```
Choose a random function h : [n] → [n] from a family of pairwise independent hash functions;
z ← 0;
while an item x arrives do
    if ρ(h(x)) > z then
        z ← ρ(h(x));
    end if
end while
Return 2^{z+1/2}
```

To improve the approximation guarantees of the AMS algorithm, we present the BJKST algorithm for approximating F_0. The BJKST algorithm uses a set to keep the sampled items. By running $\Theta(\log(1/\delta))$ independent copies in parallel and returning the medium of these outputs, the BJKST algorithm (ε, δ)-approximates the F_0-norm of the multiset S. See Algorithm 4 for formal description. [Method]

Algorithm 4 The BJKST algorithm for approximating F_0

```
Choose a pairwise independent hash function h : [n] → [n]
z ← 0                                          ▷ z is the index of the current level
B ← ∅                                          ▷ Set B keeps sampled items
while an item x arrives do
    if ρ(h(x)) ≥ z then
        B ← B ∪ {(x, ρ(h(x)))}
        while |B| ≥ 100/ε^2 do                 ▷ Set B becomes full
            z ← z + 1                          ▷ Increase the level
            shrink B by removing all (x, ρ(h(x))) with ρ(h(x)) < z
        end while
    end if
end while
Return |B| · 2^z
```

Algorithm for approximating F_2. We show that how 4-wise independent hash functions can be used to approximate F_2, and the algorithm is described in Algorithm 5. To analyse the algorithm's space complexity we prove that $\mathbf{E}[Z] = F_2$ and $\mathbf{V}[Z] \leq 2F_2^2$. These results together show that $O((1/\varepsilon^2)\log n)$ bits are sufficient to obtain a $(1+\varepsilon)$-approximation of F_2. [Proof]

The Count-Min sketch. The CM sketch is a table C of $d = \lceil \log(1/\delta) \rceil$ rows and $w = \lceil e/\varepsilon \rceil$ columns, and every row j is associated with a universal hash function $h_j : [N] \to [w]$. To maintain the CM sketch, the algorithm sets $C[j, h_j(x)] = C[j, h_j(x)] + 1$ for any $1 \leq j \leq d$ if Insert(x) arrives; the algorithm sets $C[j, h_j(x)] = C[j, h_j(x)] - 1$ for any $1 \leq j \leq d$ if Delete(x) arrives. When Query(x) arrives, the algorithm returns $m'_x = \min_{1 \leq i \leq d} C[j, h_j(x)]$. We prove that the [Method] estimate m'_x satisfies $m'_x \geq m_x$, and with probability at least $1 - \delta$ it holds that $m'_x \leq m_x + \varepsilon \cdot F_1$, where F_1 is the first moment of the multiset S. [Proof]

Algorithm 5 The BJKST Algorithm

```
Choose a 4-wise independent hash function h : [n] → [−1, 1]
y ← 0
while an item (x, ±) arrives do
    if x is inserted then y = y + h(x)
    else
        y = y − h(x)                          ▷ The case where x is deleted
    end if
end while
Return Z = y^2
```

Chernoff bound. Many of our analysis is based on the use of Chernoff bound stated as follows: Let $X_1, \ldots, X_n$ be independent random variables with $\mathbf{Pr}[X_i = 1] = \mathbf{Pr}[X_i = -1] = 1/2$. Let $X := \sum_{i=1}^{n} X_i$. Then for any $\lambda > 0$, $\mathbf{Pr}[X \geq \lambda] \leq \mathrm{e}^{-\lambda^2/(2n)}$. [Statement]

5 Graphs versus Matrices

Let $G(V, E, w)$ be a graph with weight function $w : E \to \mathbb{R}_{\geq 0}$. Let $D \in \mathbb{R}^{n \times n}$ be the diagonal matrix where $D_{u,u} = d_u$ for any vertex u, where $d_u = \sum_{u \sim v} w(u, v)$. The adjacency matrix of graph G is the matrix A defined by $A_{u,v} = w(u, v)$ if $u \sim v$, and $A_{u,v} = 0$ otherwise. The Laplacian matrix of G is defined by $L = D - A$, where A is the adjacency matrix of G. The normalised Laplacian matrix of G is defined by

$$\mathcal{L} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2}.$$

For matrix $\mathcal{L}$, we denote its n eigenvalues with $\lambda_1 \leq \ldots \leq \lambda_n$ with corresponding orthonormal eigenvectors $f_1, \ldots, f_n$. The set of n eigenvalues $\{\lambda_i\}_{i=1}^{n}$ together with their multiplicities is called the spectrum of G. [Definition]

It is easy to prove that $\lambda_1(\mathcal{L}) = 0$ with the associated eigenvector $D^{1/2} \cdot \mathbf{1}$. More graph properties can be characterised with respect to other eigenvalues, see the list below for example. [Proof]

- It holds that
$$\lambda_2 = \min_{f \perp D\mathbf{1}} \frac{\sum_{u \sim v} (f_u - f_v)^2}{\sum_u d_u \cdot f_u^2}.$$ [Proof]
- $\lambda_2 \leq n/(n-1)$, and $\lambda_2 = n/(n-1)$ iff G is a complete graph. [Proof]
- If G is not a complete graph, then $\lambda_2 \leq 1$ [Proof]

Expander mixing lemma. Let $\lambda = \max_{i \geq 2} |1 - \lambda_i|$ the spectral expansion of G. Then, the expander mixing lemma states that, for any $X, Y \subset V$, we have

$$\left| |E(X, Y)| - \frac{\mathrm{vol}X \cdot \mathrm{vol}Y}{\mathrm{vol}G} \right| \leq \lambda \cdot \sqrt{\mathrm{vol}X \mathrm{vol}Y}.$$

Notice that $\mathrm{vol}X \cdot \mathrm{vol}Y / \mathrm{vol}G$ is the expected value of $|E(X, Y)|$ in a random graph of edge density $\mathrm{vol}G/n^2$. Hence, the left side of the equation above is the difference between $|E(X, Y)|$ and its expected value in a random graph. So, a smaller value of λ shows that G is more close to be a random graph. The expander mixing lemma has several interesting applications in approximating some graph parameters, whose exact computation is known to be NP-hard: [Proof]

- The volume of any independent set in G is at most $\lambda \cdot \mathrm{vol}G$. [Proof]
- Let G be a regular graph. Then the chromatic number of G is at least $1/\lambda$. [Proof]

Graph conductance and the Cheeger inequality. For any $G = (V, E, w)$ with weight function $w : E \to \mathbb{R}$ and $S \subset V$, the conductance of S is defined by

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

where $\mathrm{vol}(S) = \sum_{u \in S} d_u$, and $w(S, V \setminus S)$ denotes the weight of edges with one endpoint in S and the other endpoint in $V \setminus S$. The **Cheeger constant** or the **conductance** of graph G is defined as

$$h_G = \min_S h_G(S).$$

Direct calculations show that

$$\lambda_2 \leq 2 \cdot h_G, \qquad \text{[Proof]}$$

and the celebrated Cheeger inequality show that h_G, which is NP-hard to compute, can be upper bounded with respect to λ_2 as well, i.e.,

$$h_G \leq \sqrt{2 \cdot \lambda_2}. \qquad \text{[Method]}$$

The core behind the proof of the Cheeger inequality is the following Algorithm 6, whose output is a set S satisfying $h_G(S) \leq \sqrt{2 \cdot \lambda_2}$.

Algorithm 6 Algorithm for finding a sparse cut

$f = D^{-1/2} f_2$
Sort all the vertices such that $f(u_1) \leq \ldots \leq f(u_n)$
$t = 0$
$S = \emptyset$
$S^\star = \{u_1\}$
while $t \leq n$ **do**
    $t = t + 1$
    $S = S \cup \{u_t\}$
    if $h_G(S) \leq h_G(S^\star)$ **then** $S^\star = S$
    end if
end while
return $S^\star$

Higher-order Cheeger Inequality. To relate the structure of multi-clusters in a graph with the other eigenvalues of $\mathcal{L}$, we define the k-way expansion of G. We call subsets of vertices (i.e. **clusters**) $A_1, \ldots, A_k$ a k-way partition of G if $A_i \cap A_j = \emptyset$ for different i and j, and $\bigcup_{i=1}^k A_i = V$. We further define the k-way expansion constant by

$$\rho(k) \triangleq \min_{\text{partition } A_1, \ldots, A_k} \max_{1 \leq i \leq k} h_G(A_i). \qquad \text{[Definition]}$$

The higher-order Cheeger inequality relates $\rho(k)$ and λ_k by the inequality

$$\frac{\lambda_k}{2} \leq \rho(k) \leq O(k^2)\sqrt{\lambda_k}. \qquad \text{[Statement]}$$

6 Spectral Clustering

Spectral clustering is one of the most popular clustering algorithms used in practice. The general framework of spectral clustering consists in (1) computing an embedding of the vertices in a low-dimensional Euclidean space using the eigenvectors of $\mathcal{L}_G$, (2) partitioning the points using a

geometric clustering algorithm, e.g., k-means, and (3) returning a corresponding partition of the graph. [Statement]

To reason about the performance of spectral clustering, we first prove that the graph G has k connected components if and only if the k smallest eigenvalues of G's normalised Laplacian matrix $\lambda_1 = \ldots = \lambda_k = 0$. Secondly, we apply the Davis-Kahan theory, which states in the graph setting that, as long as not too many edges between different clusters are added, the bottom k eigenvectors do not change too much. Together with the higher-order Cheeger inequality, we know that, as long as there is a large gap between λ_k and λ_{k+1}, spectral clustering is able to approximately recover the k clusters. Our reasoning also shows that why the smallest value of k for which there is a gap between λ_k and λ_{k+1} is a right indication for the number of clusters. [Proof] [Method]

We also study the applications of spectral clustering in image segmentation through the construction of similarity graphs, and mention the "impossibility theorem for clustering". [Method]