Chapter 1

Introduction

[This chapter is under construction!]

Clustering is a fundamental unsupervised learning tool. It can be used to search data for hidden structure, to summarize data and to classify new unseen data according to prior findings. Since applications are manifold, clustering also offers a rich variety of algorithmic problems. In this lecture, we will study clustering problems through the lens of algorithm theory. Thus, we are equipped with a mathematical formulation that allows us to judge the quality of a clustering, and our goal is to find a clustering that has the best possible quality.

The clustering problems studied in this lecture share some properties: Most of them are *partitional* clustering problems, i.e., we get some input data and want to partition it into disjoint subsets of associated elements. Furthermore, most problems that we study are *center-based*, which means that we do not only want the partitioning, but we want a set of *centers* and an assignment of points to centers which then implicitly defines the partitioning. All clustering problems are *geometric* clustering problems, which means that we have geometric information about the input data (which we then call input *points*), i.e., we know the distances between any two points.

For our study of geometric clustering problems, we need to recall the definition of *metric spaces*. The most known metric space is the Euclidean space $\mathbb{R}^d$, where two points $x, y \in \mathbb{R}^d$ with $x = (x_1, \dots, x_d)^T$ and $y = (y_1, \dots, y_d)^T$ have distance $||x - y|| = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$. It is a special case of the following more general definition.

Definition 1. *Let X be any non-empty set and $d : X \times X \to \mathbb{R}$ be a function on X. Then d is* a metric on X *if it holds for all $x, y, z \in X$ that*

1. *$d(x, y) \geq 0$ and $d(x, y) = 0$ if and only if $x = y$,*
2. *$d(x, y) = d(y, x)$, and*
3. *$d(x, y) \leq d(x, z) + d(z, y)$.*

We call a pair (X, d) consisting of a finite set X and a metric d a metric space.

We refer to Condition 3 in Definition 1 as the *triangle inequality*. Usually we call elements of X *points*. Furthermore, we extend any metric d to finite point sets by setting

$$d(x, A) = \inf_{y \in A} d(x, y)$$

for any finite $A \subseteq X$ and

$$d(A, B) = \inf_{x \in A, y \in B} d(x, y).$$

for any finite $A, B \subseteq X$.

The Euclidean space $\mathbb{R}^d$ with the Euclidean distance is one example for a metric space. We will encounter it later in the lecture when we study the k-means problem. This modeling is common in machine learning and statistics, where k-means comes up as an important problem. However, clustering is also a classical topic in graph theory. Here, it is more common to study the case of *finite metric spaces*. The input then consists of a *finite* point set P and a distance function $d : P \times P \to \mathbb{R}$ which is metric, i.e. (P, d) form a metric space. In this scenario, d simply consists of $|P|^2$ numbers that give the distance for every pair of points in P. The can thus be viewed as an undirected complete weighted graph: the vertices are the points in P, and the weight of edge $\{x, y\}$ is $d(x, y) = d(y, x)$ for all $x, y \in P$.

In this context, the the metric space (P, d) can also be implicitly given by a *graph metric*: For any (not necessarily complete) graph $G = (V, E)$ weighted by a function $w : V \times V \to \mathbb{R}$, we define the *shortest path distance* $d_s(x, y)$ between $x, y \in V$ to be the length of the shortest path between x and y in G. One can show that d_s is a metric on V, and thus G and w implicitly define the metric space (V, d_s).

All problems studied in this lecture are minimization problems. As it turns out, for most of them, it is not known how to compute optimal solutions in polynomial time, and it may well be impossible to do so, since the problems are NP-hard. For this reason, we will resort to approximation algorithms that find a reasonably good solution instead of an optimal one.

Recall that an α-approximation algorithm for a minimization problem is an algorithm that outputs solutions with a value that is at most α times the value of an optimal solution. Let $\mathfrak{I}$ be the set of all possible input instances to an optimization problem, let ALG be an algorithm that computes a feasible solution $S^{ALG}(I)$ for any $I \in \mathfrak{I}$. For any $I \in \mathfrak{I}$, let $S^*(I)$ be an optimal solution for I. For a minimization problem, suppose that $c(S)$ is the cost of a solution S. Then ALG is an *α-approximation algorithm* if

$$\sup_{I \in \mathfrak{I}} \frac{c(S^{ALG}(I))}{c(S^*(I))} \leq \alpha.$$

Whenever we talk about α-approximation algorithms, we implicitly mean algorithms that have a polynomial worst-case running time. All problems considered in this lecture can be solved optimally given exponential running time.

Our studies will also include *randomized* approximation algorithms. For randomized algorithms, $c(S^{ALG}(I))$ or $v(S^{ALG}(I))$, respectively, is a random variable. We might either want to achieve that it is close to the optimum value with high probability, or that its expected value is close to the optimum value. We choose the second alternative. Thus, we say that a randomized algorithm ALG is an *α-approximation algorithm in expectation* for a minimization problem if

$$\sup_{I \in \mathfrak{I}} \frac{\mathbf{E}\left[c(S^{ALG}(I))\right]}{c(S^*(I))} \leq \alpha.$$

As for deterministic algorithms, we also require randomized approximation algorithms to have a polynomial worst-case running time.

1.1 Mathematical facts

This section collects mathematical facts that we need throughout the lecture.

Fact 2. *For all $r \in \mathbb{R}$ with $r \neq 1$ and all $i \geq 0$,*

$$\sum_{i=0}^{n} r^i = \frac{1 - r^{n+1}}{1 - r}.$$

If $|r| < 1$, then

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1 - r}.$$

Chapter 2

The happy world of k-center

We start with the study of the *k-center problem.* It is a classical problem from graph theory, and it is one of the rare problems where we precisely know the best possible approximation ratio under the assumption that $P \neq NP$. This approximation ratio can be achieved by two elegant approximation algorithms.

2.1 The metric k-center problem

The input to the k-center problem is a finite metric space (P, d) and a number $k \in \mathbb{N}^{\geq 1}$. The k-center problem is a center-based clustering problem, meaning that we are searching for a set of centers C that give a good clustering when every point is assigned to the center closest to it. Center-based clustering problems differ in the evaluation of the quality of a center set. The k-center problem is a so-called min-max problem: We are minimizing the maximum distance of any point to its closest center.

Definition 3. *Let (P, d) be a metric space and $k \in \mathbb{N}^{\geq 1}$ be a number. The k-*center

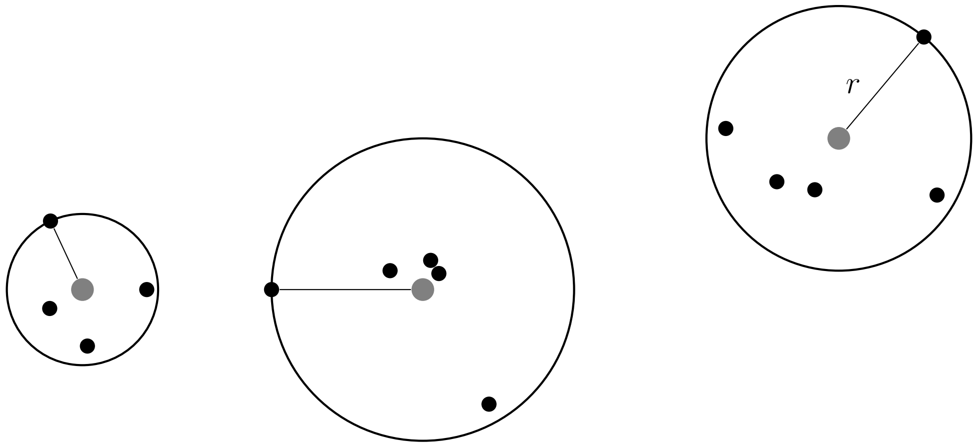

Figure 2.1: An example for a k-center instance and solution with $k = 3$.

problem *is to compute a set $C \subseteq P$ with $|C| = k$ that minimizes*

$$\max_{x \in P} \min_{c \in C} d(x, c).$$

We call the value $\max_{x\in P} \min_{c\in C} d(x, c)$ either *k-center objective value* or *maximum radius* of the solution C. Figure 2.1 shows a point set, three chosen centers and the radii of the resulting three clusters. The maximum radius of the depicted solution is labeled by r.

2.2 A simple and elegant approximation

Gonzalez' algorithm [Gon85] gives a greedy 2-approximation for the k-center problem. The algorithm starts by picking an arbitrary point $c_1 \in P$ as the first center. Then it does $k - 1$ iterations of the following: Choose the point c_i which is farthest away from the previously chosen centers $\{c_1, \ldots, c_i\}$ to be the center. This is repeated until k centers are chosen.

```
farthest-first-traversal(a finite metric space $(P, d)$, $k \in \mathbb{N}^{\geq 1}$)
  1. Choose a center $c_1$ arbitrarily from $P$ and set $C^1 = \{c_1\}$
  2. For $i = 2$ to $k$ do
  3.     Choose $c_i$ as (one of) the points maximizing $d(x, C^{i-1})$,
         i.e., $c_i = \arg\max\{d(x, C^{i-1}) \mid x \in P\}$
  4.     Set $C^i = C^{i-1} \cup \{c_i\}$
  5. Return $C = \{c_1, \ldots, c_k\}$
```

Figure 2.2 shows an example run of Algorithm `farthest-first-traversal` with $k = 3$. The first center is chosen arbitrarily. Center 2 and 3 are then chosen as points with maximum distance to the previously chosen centers. This means that the algorithm has the tendency to choose points that are at the border of a cluster. The resulting solution is worse than the solution depicted in 2.1, but its maximum radius r' only slightly larger than r from the previous solution. The following theorem shows that in general, the maximum radius is at most twice the maximum radius of an optimal solution.

Theorem 4. *Algorithm `farthest-first-traversal`$((P, d), k)$ computes a 2-approximation for the k-center problem.*

Proof. We set $\Delta := \max\{d(x, C) \mid x \in P\}$, this is the objective value of the solution computed by `farthest-first-traversal`$((P, d), k)$. Then there is a point $y \in P$ that actually has distance Δ to its closest center in C, i.e., $d(y, C) = \min_{c\in C^k} d(y, c) = \Delta$. Now we observe that the distance between y and its closest center can only get shorter during the execution of the algorithm, i.e., $d(y, C^i) \geq d(y, C^k)$ for all $i \in \{1, \ldots, k\}$. Thus, $d(y, C^i) \geq \Delta$ is true for all $i \in \{1, \ldots, k\}$. Since Gonzalez' algorithm

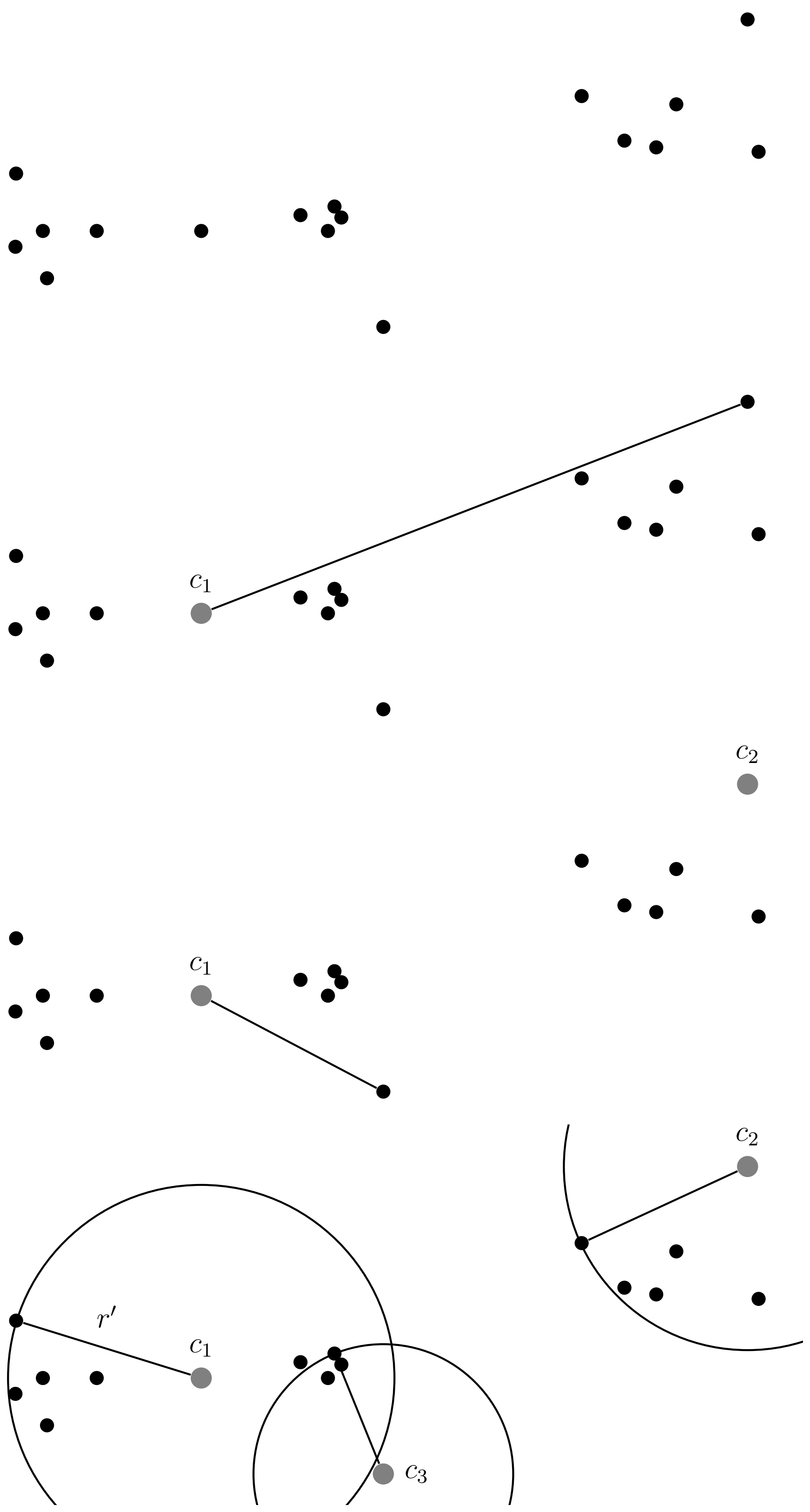

Figure 2.2: An example run of farthest first traversal with $k = 3$.

always chooses the point that has maximum distance to the previously chosen centers, $d(y, C^i) \geq \Delta$ implies $d(c_{i+1}, C^i) \geq \Delta$ for all $i \in \{1, \ldots, k-1\}$. This means that the centers and y form a set of $k+1$ points with pairwise distance Δ. Any solution has to have a cluster with two of these points, and this cluster then has at least radius $\Delta/2$. Since the optimal solution has a maximum radius of at least $\Delta/2$, and the computed solution has a radius of Δ, we conclude that the computed solution is a 2-approximation. □

Lemma 5. *Algorithm* `farthest-first-traversal`(P, d) *can be implemented to run in time $\mathcal{O}(|P| \cdot k)$.*

Proof. Line 1 takes constant time. The loop in lines 2-4 is executed $k-1$ times. For line 3, the algorithm needs to find the point whose closest center is farthest away. To keep track of this, we use an array $c_{\min}$ which stores the closest center (so far) for each point, i.e. we preserve the equality $c_{\min}(x) = \arg\min\{c \in C^{i-1} d(x, c)\}$ for all $x \in P$. The array is initialized in time $\mathcal{O}(|P|)$ before starting the loop. In the loop, when we choose a new center c_i, we compare $d(x, c_i)$ with $d(x, c_{\min}(x))$ and update $c_{\min}(x)$ accordingly. This takes constant time for every point, i.e., updating $c_{\min}$ takes time $\mathcal{O}(|P|)$. We can find the point with the maximum $c_{\min}$ value simultaneously or find it in another $\mathcal{O}(|P|)$ operations after the updating is finished. Thus, lines 2-4 take time $\mathcal{O}(|P|\dot{k})$. □

Was part of lecture 1, but is not used until later

The following definitions will later be part of the respective chapters.

Definition 6. *Let (P, d) be a metric space and $k \in \mathbb{N}^{\geq 1}$ be a number. The k-*median problem *is to compute a set $C \subseteq P$ with $|C| = k$ that minimizes*

$$\sum_{x \in P} \min_{c \in C} d(x, c).$$

Definition 7. *Let (P, d) be a metric space and $f : P \to \mathbb{N}^{\geq 1}$ be a function of* opening costs*. The* facility location problem *is to compute a set $C \subseteq P$ that minimizes*

$$\sum_{x \in P} \min_{c \in C} d(x, c) + \sum_{c \in C} f(c).$$

Definition 8. *Let $P \subset \mathbb{R}^d$ be a finite set of points from the Euclidean space and $k \in \mathbb{N}^{\geq 1}$ be a number. The k-*means problem *is to compute a set $C \subseteq P$ with $|C| = k$ that minimizes*

$$\sum_{x \in P} \min_{c \in C} ||x - c||^2.$$

– end of lecture 1 –