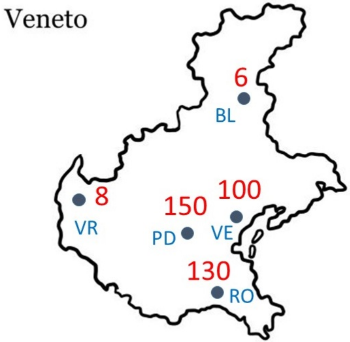
HOMEWORK 2

k-center with outliers: motivation



Red numbers = potential customers

If you must open 3 stores, where would you open them?

If you used k-center (with k=3) to select the store locations, what would be the result?

k-center with outliers: problem

Input Set P of N points from (M, d), integers $k, z \in [1, N)$

Output Set $S \subset P$ of k centers which minimize

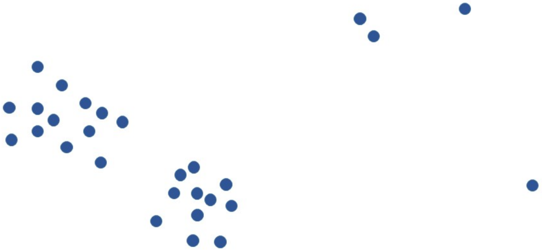
$$\Phi_{\text{kcenter}}(P - Z_S, S), = \max_{x \in P - Z_S} d(x, S)$$

where $Z_S = z$ points of P farthest from S.

Remarks:

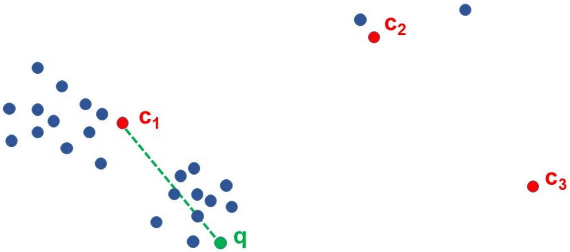
- It is the k-center problem where the objective function is allowed to disregard the distances of the z points farthest from the center. For noisy datasets, this yields a better center selection.
- The objective function is uniquely determined by P and S
- with $z = 0$ we have the standard k-center problem

Example



Input P

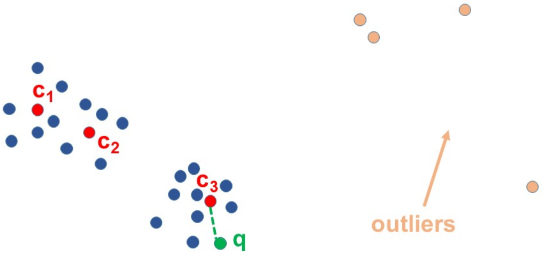
Example



Optimal solution with k=3 centers

Value of objective function = $d(c_1,q)$

Example



Optimal solution with k=3 centers and z=4 outliers

Value of objective function = $d(c_3,q)$

typically much less than the value of the objective function with $z=0$

k-center with outliers: weighted variant

Input Set $(P, w(\cdot))$ of N weighted points from (M, d), integers $k, z \in [1, N)$

Output Set $S \subset P$ of k centers which minimize

$$\Phi_{\mathrm{kcenter}}(P - Z_S, S), = \max_{x \in P - Z_S} d(x, S)$$

where

$$Z_S = \text{largest set of points farthest from } S \text{ of total weight} \leq z$$

Remark: if we consider P sorted by distance from S, then Z_S is the longest suffix of points, in the sorted sequence, of total weight $\leq z$.

Example

	Points sorted by distance from S											
distance	0	0	0	0	1.3	1.6	1.9	2.4	2.5	2.7	4.1	6.3
weight	9	11	6	4	13	2	12	3	4	2	1	2

$z = 3, 4$

value of obj. function with $z = 3, 4$

$z = 9, 10, 11$

value of obj. function with $z = 9, 10, 11$

k-center with outliers: algorithm kcenterOUT

For a radius $r > 0$, a set $Z \subset P$, and a point x, define the ball of Z with radius r centered at x as

$$B_Z(x, r) = \{y \in Z \ : \ d(x, y) \leq r\}.$$

Overview of the algorithm:

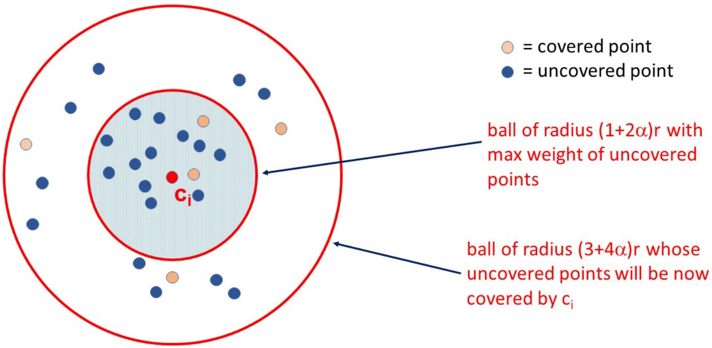
- Geometric sequence of guesses $r = r_{min} \cdot 2^j$ for $j = 0, 1, \ldots$, where r_{min} is a lower bound to the optimal objective function.
- S = current centers, Z = *uncovered points*, α = parameter.
- For each guess r, initialize $S = \emptyset$ and $Z = P$, and execute k iterations. In the i-th iteration:

  ith center added to S

  - add to S the point $c_i \in P$ which maximizes the total weight in $B_Z(c_i, (1 + 2\alpha)r)$.
  - remove from Z the points of $B_Z(c_i, (3 + 4\alpha)r)$
- The algorithm stops at the smallest guess r for which the points of Z (i.e., the *outliers*), have total weight $\leq z$, and returns S

when the total weight of the points of Z is > z ⇒ proceed with the next guess 2·r

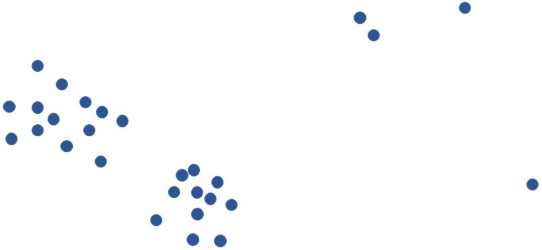
Selection of *i*-th center



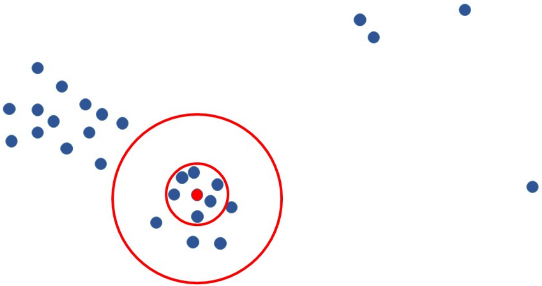
kcenterOUT(P, k, z, α): pseudocode

```
r ← (min distance between first k+z+1 points)/2;  // r_min
while (true) do
    Z ← P;  S ← ∅;  W_Z = Σ_{x∈P} w(x);
    while ((|S| < k) AND (W_Z > 0)) do
        max ← 0;
        foreach x ∈ P do
            ball-weight ← Σ_{y∈B_Z(x,(1+2α)r)} w(y);
            if (ball-weight > max) then
                max ← ball-weight;
                newcenter ← x;
        S ← S ∪ {newcenter};
        foreach (y ∈ B_Z(newcenter, (3+4α)r)) do
            remove y from Z;
            subtract w(y) from W_Z;
    if (W_Z ≤ z) then return S;
    else r ← 2r;
```

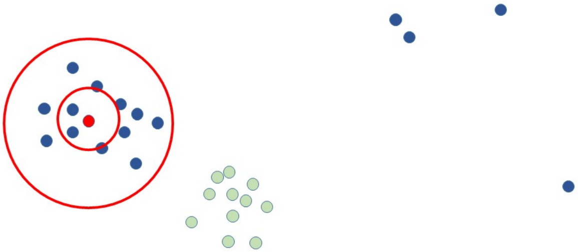
Example: $k = 2$, $z = 4$, unit weights



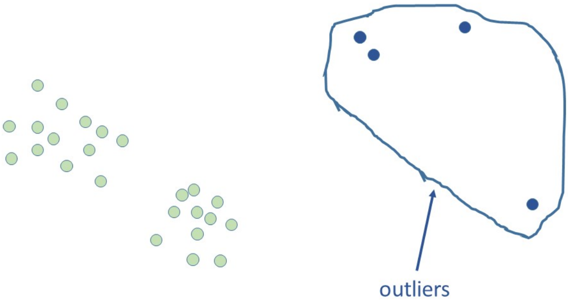
Example: $k = 2$, $z = 4$, unit weights



Example: $k = 2$, $z = 4$, unit weights



Example: $k = 2$, $z = 4$, unit weights



Quality of solution

- The value of the objective function for the returned solution S is at most $(3+4\alpha)r$, where r is the last guess. However, it can be smaller.
- If `kcenterOUT` is executed on the entire dataset with unit weights and $\alpha = 0$, then it returns a 3-approximation.
- If `kcenterOUT` is executed on a weighted coreset $T \subseteq P$ obtained by using $k+z$-center on P (or on each of L partitions of P), with $\alpha = 2$, then it returns a 13-approximation.

Homework 2

Write a sequential program that:

1. Contains a method that implements sequentially `kcenterOUT` (weighted variant and Euclidean points!)
2. Reads P, k, z input parameters and runs `kcenterOUT`$(P, k, z, \alpha = 0)$ using unit weights.
3. Computes the value of the objective function.
4. Reports several statistics.

Detailed specification in Moodle Exam

Homework 2: hints

Each point is an instance of

- Representation of points (arbitrary dimensionality):
  - **Java:** class `org.apache.spark.mllib.linalg.Vector`, which can be manipulated through static methods offered by class `org.apache.spark.mllib.linalg.Vectors`.
  - **Python:** tuple of float
- You will not use Spark (except for the points representation in Java)
- Distance computations are the bottleneck (especially in Python!). You may precompute all distances: up to 10^4 they are likely to fit it RAM, and for the MapReduce implementation (Homework 3) we won't need to run it on larger instances.
- Carefully select the representation for Z.
- Debug your code for correctness and efficiency (try to identify sources of inefficiency!)

Homework 3

In Homework 3 you will implement a 2-round MR-algorithm for k-center with outliers, where Round 1 computes a weighted coreset of size $(k + z) \cdot L$ using Farthest-First Traversal in each of L partitions, and Round 2 runs `kcenterOUT` on the weighted coreset.

We will provide a template for the homework and you must:

1. write the code for FFT (Round 1);
2. add code for `kcenterOUT` (Round 2), recycled from Homework 2;
3. run the algorithm on CloudVeneto, assessing running time, scalability with respect to parallelism, and quality of solution.

Details will be given soon.