Algorithms for Big Data Fall 2020

Lecture 17 & 18 — November 9 & 13, 2020

Prof. Cyrus Rashtchian *Topics: Clustering Intro*

Overview. We introduce clustering, providing several examples of objectives. We also provide an algorithm for k-center clustering, which is a 2-approximation.

1 Clustering

Clustering means dividing data into groups based on similarity. Elements within the same group should be more similar to each other than to any element of another group. A partition of the dataset into non-empty subsets will serve as our definition of a clustering. Similarity joins can be thought of as a form of clustering To analyze a clustering algorithm, we need to measure the quality of a clustering. There are many possible objectives

- k-center for any metric space $(\mathcal{X}, d_{\mathcal{X}})$. Given a dataset $S \subseteq \mathcal{X}$, the goal is to quickly find a set of centers $T \subseteq \mathcal{X}$ with the constraint $|T| = k$. The objective is that $\max_{a \in S} d_{\mathcal{X}}(a, T)$ is minimized over all such sets T. Here we extend the definition of a distance function to sets by $d_{\mathcal{X}}(a, T) = \min_{b \in T} d_{\mathcal{X}}(a, b)$.
- k-median: choose k cluster centers (not necessarily part of the input data), and minimize the ℓ_1 distance to the cluster centers.
- k-means [Mac67, Ste56]. choose k cluster centers (not necessarily part of the input data), and minimize the ℓ_2 distance to the cluster centers (sometimes the square of the ℓ_2 distance is more convenient).
- Connected components: when the input is a graph, sometimes it is already interesting to just find the connected subgraphs. This is relevant, for example, when you build the similarity graph (connect pairs with distance $\leq r$), and when this graph is disconnected because the clusters are separated.
- Sparsest cut and Spectral clustering: minimize certain graph-cut objectives.
- Correlation clustering: If we are clustering vertices of a graph, then we could maximize the number of edges within each cluster and minimize the number of edges between clusters.
- Mixture models: If the dataset is sampled from an underlying distribution, then the goal could be to find a clustering that helps us estimate certain parameters (*e.g.,* determining means and variances of a mixture of Gaussians).
- Hierarchical clustering. As one example of an agglomerative method, we mention *linkage-based* algorithms. Here, the clustering is initialized with every point being its own separate cluster. Then, in each iteration, the two closest clusters are merged. Variations exist depending on the measure of cluster closeness. Some popular examples include single-linkage (distance between the two closest points in two clusters) and average-linkage (mean distance between points in two clusters). In practice, these criteria may lead to very different clusterings.

We will mostly focus on k-center and k-means (which are top-down/divisive). However we provide a brief glimpse into hierarchical clustering, which is an important agglomerative clustering framework.

1.1 Linkage-based Hierarchical Clustering

Later, we present Gonzalez's algorithm, where clusters can be formed by associating each point in $\mathcal{X}$ to its closest center in T. Moreover, this can be viewed as a *top-down* approach, since we are broadly grouping related points based on common criteria. In contrast, clusters could also be formed in a *bottom-up* fashion, where nearby points are iteratively grouped together until clusters emerge. Clustering algorithms are often broadly organized into two categories based on whether they operate in a top-down or bottom-up manner, known as *divisive* versus *agglomerative* methods in the clustering literature [FHT01, HMMR15, KR09].

As an example of some bottom-up approaches, we consider linkage-based methods. In all of them, the points in $\mathcal{X}$ start as singleton clusters, so there are n different clusters. Then, at each step we *the closest pair of clusters*. There are many different definitions. Let's say that $\mathcal{C}$ is a set of clusters (a partition of $\mathcal{X}$). In the beginning, $|\mathcal{C}| = n$, and then, as the clusters merge there will be fewer and fewer clusters.

Popular merging rules:

- **Single-linkage:** merge the pair of clusters C, C' that minimizes
$$\min_{x \in C, y \in C'} d(x, y)$$
- **Average-linkage:** merge the pair of clusters C, C' that minimizes
$$\frac{1}{|C| \cdot |C'|} \sum_{x \in C, y \in C'} d(x, y)$$
- **Complete-linkage:** merge the pair of clusters C, C' that minimizes
$$\max_{x \in C, y \in C'} d(x, y)$$

Each of these merge rules will lead to a different overall clustering. There are two main stopping criteria. Either you can stop when where are k clusters in total. Or, you can keep going to the end and get a full hierarchy of the points.

2 k-Center Clustering

An example objective is the k-center problem, which is well-defined for any metric space with distance $d(\cdot, \cdot)$. Given a dataset $S \subseteq \mathcal{X}$, the goal is to quickly find a set of centers $T \subseteq \mathcal{X}$ with the constraint $|T| = k$. The objective is that $\max_{x \in \mathcal{X}} d(x, T)$ is minimized over all such sets T. As before, we extend the definition of a distance function to sets by $d_{\mathcal{X}}(x, T) = \min_{c \in T} d_{\mathcal{X}}(x, c)$.

Convenient Notation. Let $T \subseteq \mathcal{X}$ be a set of cluster centers. This leads to a clustering by associating each point $x \in \mathcal{X}$ to the closest cluster center $c \in T$.

If $|T| = k$, then we can define S_T as a vector which the distances to the closest center $c \in T$. More precisely, we define the n-dimensional vector, where we denote $S = \{x_1, x_2, \ldots, x_n\}$. Then,
$$S_T = [d(x_1, T), d(x_2, T), \ldots, d(x_n, T)],$$
which is a vector in $\mathbb{R}^n$ with nonnegative entries (because distances are nonnegative). The ith coordinate is the distance between x_i and the closest point in T. Then, many objectives can be viewed as norms of this $\mathcal{X}_T$ vector, where we want to find $T \subseteq \mathcal{X}$ such that

- k**-center:** minimize $\text{cost}(T) = \|S_T\|_\infty$.
- k**-median:** minimize $\text{cost}(T) = \|S_T\|_1$.
- k**-means:** minimize $\text{cost}(T) = \|S_T\|_2^2$.

This makes it very clear that the only difference in these different objectives is how to measure the overall cost (which is based on different norms). It's also easiest to see in this notation that the points $\mathcal{X}$ do not have to be real vectors; they can be in any metric space. However, it's very common to consider k-median and k-means when $S \subseteq \mathbb{R}^d$ is a set of real vectors, and the distance $d(\cdot,\cdot)$ is the ℓ_1 or ℓ_2 distance as well. This also makes it easier to talk about the 'center' or the 'centroid' of a cluster, which can be defined as the center of mass of the points in the cluster.

It's a big debate how to choose the number of clusters k correctly. No one really knows. If k is too small, you get spread out clusters. If k is too large, you may not capture the structure of the data. Also, if $k = n$, then the cost is always zero (because if $S = T$, then the closest center to a point can be taken to be point itself). One benefit of bottom-up agglomerative methods is that you do not have to specify k ahead of time.

2.1 The greedy "farthest point" traversal algorithm

For this section, we are going to use $\mathcal{X}$ to denote the dataset as well as the metric space. The reason is that for an arbitrary metric space, and for the k-center objective, we don't have information about the metric space outside of the dataset (unlike for real vectors, where you know the whole space). You can think of the dataset as a subset of a larger space, but we won't use points outside of $S = \mathcal{X}$.

Gonzalez showed that this problem is NP-hard in general, and he exhibited a simple 2-approximation algorithm [Gon85]. His algorithm greedily finds points in $\mathcal{X}$ that are farthest from previously chosen centers, and it runs in time $O(k|\mathcal{X}|) = O(kn)$. More precisely, the algorithm iteratively constructs the set of centers T by adding the farthest point in $\mathcal{X}$ to the set of centers so far, and then repeats until there are $|T| = k$ centers. This algorithm is nice because it provably works for any metric space. Note that when k is relatively small compared to $|\mathcal{X}| = n$, this works well, taking time $\Theta(kn)$. However, when $k = \Omega(n)$, then this algorithm takes quadratic time, $\Theta(n^2)$. In pseudo-code, we have the following algorithm.

Farthest-First Traversal Algorithm.

```
Choose any $z \in \mathcal{X}$ and initialize $T_1 = \{z\}$
For $i = 1, 2, 3, \ldots, k-1$ :
    $z = \arg\max_{x \in \mathcal{X}} d(x, T_i)$
    $T_{i+1} \leftarrow T_i \cup \{z\}$
Output $T \leftarrow T_k$.
```

In each step, we added the farthest point in $\mathcal{X}$ to the current set of centers. Notice that we never add the same point twice because $d(x,T) = 0$ if $x \in T$ by definition.

As an aside, this traversal actually has other uses in geometric algorithms, and you can prove some nice properties about it. Moreover, it can be used to initialize other algorithms. In fact, some k-means implementations first do the Farthest-First Traversal to get a good candidate set of k centers. Then, local heuristics are used to improve the clustering quality.

Analysis. The time of the Farthest-First Traversal Algorithm is $O(kn)$ when $n = |\mathcal{X}|$. In each step, we needed to find the farthest point in $\mathcal{X}$ from the current set of clusters. Naively this would take $O(n)$ time for each point in the current set of centers, leading to an overall running time of $k \cdot O(kn)$. However, we can avoid some redundant work. Assume we have the distances $d(x, T_i)$ for all $x \in \mathcal{X}$. The maximum such distance tells us which point to add. Furthermore, we can update for $T_{i+1} = T_i \cup \{z\}$ using the identity

$$d(x, T_{i+1}) = \min\{d(x, z),\ d(x, T_i)\}.$$

In the 'convenient notation' from above, it takes $O(n)$ time to compute $\mathcal{X}_{T_{i+1}}$ when you know $\mathcal{X}_{T_i}$. So, we only update $O(n)$ distances at each of the k steps, and we can find the farthest point.

We prove that the cost of this algorithm's solution is at most a factor of two from the optimal. We use T^* to denote the optimal set of center. In other words, T^* is the true minimizer of $\text{cost}(T) = \|\mathcal{X}_T\|_\infty$ over all sets $T \subseteq \mathcal{X}$ with $|T| = k$.

Theorem 1. *If T is the solution output by the Farthest-First Traversal Algorithm, and T^* is the optimal solution, then*

$$\text{cost}(T) \leq 2 \cdot \text{cost}(T^*)$$

Proof. Let

$$r = \text{cost}(T) = \max_{x \in \mathcal{X}} d(x, T)$$

be the cost of T, and let x be a the point achieving this maximum. Then, consider $T \cup \{x\}$, which is a set of $k+1$ points, and which has the property that every pair of points has distance at least r by the construction of T. By the pigeonhole principle, using that $|T^*| = k$, some two points in $T \cup \{x\}$ must have the same closest point in T^*. Therefore, as these two points have distance $\geq r$, the distance of one of these points to T^* must be at least $r/2$. In other words,

$$\text{cost}(T^*) \geq r/2 = \frac{1}{2}\text{cost}(T),$$

which is what we wanted to show. □

Perhaps surprisingly, it is NP-Hard to do better. More precisely, for any constant $\varepsilon > 0$, it is NP-Hard to find a k-center solution that is a $(2 - \varepsilon)$ approximation algorithm in the worst-case.

2.2 Second Proof

We also provide a second proof of the above theorem, which has a slightly different flavor.

Alternative Proof of Theorem 1. Recall that a set of centers T defines a clustering $\mathcal{C}$ by associating each point in $\mathcal{X}$ with its closest center. Let $\mathcal{C}^*$ be the optimal clustering based on T^* the optimal set of centers. If every cluster in $\mathcal{C}^*$ contains exactly one point in T, then the claim follows, because the distance to the center in T is at most twice the distance to the center in T^* by the triangle inequality. To see this, you can go from any x to $c^* \in T^*$ to $c \in T$, since

$$d(x, c) \leq d(x, c^*) + d(c, c^*) = d(x, T^*) + d(c, T^*) \leq 2 \cdot \text{cost}(T^*).$$

Otherwise, by the pigeonhole principle, there must be two centers $c, c' \in T$ that are both in the same cluster in $\mathcal{C}^*$, with center $c^* \in T^*$. Assume WLOG that c was added to T before c' was added by the Farther First Traversal Algorithm. For notation, say that c was added in step i so that

$T_{i+1} = T_i \cup \{c'\}$ and $c \in T_i$ already. Since the algorithm always adds the furthest point, we have that

$$\text{cost}(T) \leq \text{cost}(T_i) = d(c', T_i) \leq d(c, c') \leq d(c, c^*) + d(c', c^*) \leq 2 \cdot \text{cost}(T^*).$$

□

3 Clustering Real Data: Speed, Filtering, and Outliers

For large datasets, clustering algorithms must be very efficient. An integral component of the running time comes from evaluating a distance function on pairs of points. To optimize an algorithm, we should reduce both the number of pairwise comparisons and the time it takes to perform a comparison. To this end, many *filtering* methods have been proposed, such as Canopy Clustering by McCallum, Nigam, and Ungar [MNU00] and Clustering Using Representatives (CURE), by Guha, Rastogi, and Shim [GRS98]. Another way to filter is to apply a hash function to the data and only compare points in the same hash bucket. LSH is also a good filtering method.

Filtering methods can be very effective when the dataset is comprised of underlying clusters that are separated in the metric space. In this case, we say that the dataset is *clusterable*, and formal definitions of clusterability use a definition of *separation* [ABDLS13, BLG14]. The simplest assumption may be that points in different clusters are strictly farther apart than same-cluster points. This can be in terms of a single scale (e.g., distance at most r vs. distance at least $2r$) or a more global criteria (e.g., misclassifying any point leads to a huge increase in the objective value) [ABC+15, BBG13, MMV14].

In real datasets, assumptions of separability or clusterability may be violated. This could be due to imprecise measurements or to noise in the data collection process. In these cases, it becomes important to understand how the output of a clustering algorithm changes if adversarial points, or *outliers*, are added to the dataset [ABDLS13, BLG14]. From a theoretical point of view, the question is whether algorithms still have good guarantees in the presence of outliers. Fortunately, in many cases, the non-outlier analysis will still hold even in the presence of outliers [ABDLS13, BLG14]. For example, there exist algorithms that provide a good approximation for the optimal k-center clustering even with many outliers [MKC+15].

3.1 Questions to Ponder

Although we don't have more homework problems, I want to list a set of questions to check your understanding and explore the topics in this lecture.

1. Design a dataset where the single-linkage hierarchical clustering algorithm will iteratively build one large cluster by merging a singleton into the larger cluster at every step (so the eventual dendrogram is a depth $n-1$ binary tree when there are n input points). For this dataset, what results are produced by average-linkage and complete-linkage?

2. One issue for hierarchical clustering in practice is that it can be prohibitively slow. Determine the running times for the three linkage-based algorithms presented above. One natural idea to speed up the algorithm is to use ideas from LSH to merge approximately similar clusters at every step, which would reduce the running time. How would you do this? How would you analyze the performance compared to computing the exact closest pair of clusters to merge?

3. Design a dataset where k-center and k-means will have (i) the same optimal clustering, or (ii) a very different clustering. You can choose a dataset $S \subseteq \mathbb{R}^d$ and use the Euclidean distance

to measure distances.

4. Explain why the 2-approximation for k-center does not lead to any constant factor approximation for k-means, even for real vectors and Euclidean distance.

References

[ABC$^+$15] Pranjal Awasthi, Afonso S Bandeira, Moses Charikar, Ravishankar Krishnaswamy, Soledad Villar, and Rachel Ward. Relax, no need to round: Integrality of clustering formulations. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 191–200. ACM, 2015.

[ABDLS13] Margareta Ackerman, Shai Ben-David, David Loker, and Sivan Sabato. Clustering Oligarchies. In *AISTATS*, 2013.

[BBG13] Maria-Florina Balcan, Avrim Blum, and Anupam Gupta. Clustering Under Approximation Stability. *Journal of the ACM (JACM)*, 60(2):8, 2013.

[BLG14] Maria-Florina Balcan, Yingyu Liang, and Pramod Gupta. Robust Hierarchical Clustering. *Journal of Machine Learning Research*, 15(1):3831–3871, 2014.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*, volume 1. Springer series in statistics New York, 2001.

[Gon85] Teofilo F Gonzalez. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38:293–306, 1985.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *ACM Sigmod Record*, volume 27, pages 73–84. ACM, 1998.

[HMMR15] Christian Hennig, Marina Meila, Fionn Murtagh, and Roberto Rocci. *Handbook of Cluster Analysis.* CRC Press, 2015.

[KR09] Leonard Kaufman and Peter J Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*, volume 344. John Wiley & Sons, 2009.

[Mac67] J. Macqueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. Mathematical Statist. Probability*, pages 281–297, 1967.

[MKC$^+$15] Gustavo Malkomes, Matt J Kusner, Wenlin Chen, Kilian Q Weinberger, and Benjamin Moseley. Fast Distributed k-center Clustering with Outliers on Massive Data. In *NIPS*, 2015.

[MMV14] Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Bilu–Linial Stable Instances of Max Cut and Minimum Multiway Cut. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 890–906. SIAM, 2014.

[MNU00] Andrew McCallum, Kamal Nigam, and Lyle H Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *SIGKDD*, pages 169–178. ACM, 2000.

[Ste56] H. Steinhaus. Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci*, 1:801–804, 1956.