K-Means

Cornell CS 4/5780

Spring 2022

In the first few lectures of this class we discussed supervised learning problems. While we will return to this setup soon, for this lecture and the next we will take a brief detour to discuss *unsupervised learning*. Colloquially, a key distinction from the supervised learning setup is that we now only have features $x_1, \dots, x_n$ (as usual, $x_i \in \mathbb{R}^d$) and no corresponding labels (in other words, no y_i). This means that in this setting we are not looking to make predictions. Instead, we are primarily interested in uncovering structure in the feature vectors themselves.

"Uncovering structure" is an admittedly vague concept and, more generally, the unsupervised learning problem tends to be more subjective then the supervised learning setup. Nevertheless, unsupervised learning is an important problem with applications such as data visualization, dimensionality reduction, grouping objects, exploratory data analysis, and more. Perhaps the most canonical example of unsupervised learning is clustering—given the n feature vectors we would like to group them into k collections based on similarity. For example, each feature vector x_i could represent a collection of movie ratings from user i and we would like to group the n users into sets of individuals with similar tastes in movies. We could then label the groups by examining the feature vectors (e.g., fans of historical dramas, animated films, or comedies), use the groupings to make recommendations, and even classify new users into existing groups. However, there is no ground truth taxonomy of movie watchers so we are never actually "predicting" a label and have no known baseline to compare with.

A key feature of the unsupervised learning problem is that the structure we find (if it exists) is intimately tied to the algorithm/methodology we choose. In this class we will primarily talk about two algorithms and, therefore, two types of structure we hope to uncover. First, we will explore how k-means clustering allows us to determine similar groups of feature vectors based on euclidean distances.[1] Second, we will see how principal component analysis allows us to determine: (1) if the collection of features vectors lies in a lower dimensional space than their latent dimension d and (2) a small set of "synthetic" features that can be used to describe variability in the data.

K-means clustering

Given a set of n data points (i.e., feature vectors) $x_1, \dots, x_n$ with $x_i \in \mathbb{R}^d$ the goal of k-means clustering is to partition the data into k groups[2] where each group contains similar objects as defined by their euclidean distances.[3] The ideal setting for such a procedure is readily illustrated in two dimensions by fig. 1.

Figure 1: Clustering data into two clearly defined clusters based on euclidean distance.

Figure 1: Clustering data into two clearly defined clusters based on euclidean distance.

Importantly, fig. 1 started with points that lend themselves to a clustering based on euclidean distance—there were two clearly distinct sets of points. However, in the general setting we do not know that such a strategy will be effective a priori. While we will always be able to run the k-means algorithm presented here, there are data sets for which the resulting partition may be meaningless, e.g., as in fig. 2.

Figure 2: An example where clustering based on euclidean distance seems ill-conceived.

Figure 2: An example where clustering based on euclidean distance seems ill-conceived.

The k-means objective, formally

To formalize the clustering problem we need to introduce a bit of notation. Specifically, let the k subsets $\mathcal{C}_1, \dots, \mathcal{C}_k$ with $\mathcal{C}_i \subseteq \{1, \dots, n\}$ specify the k clusters of points. In other words, if x_i is assigned to cluster j then $i \in \mathcal{C}_j$. To enforce that these subsets define a proper partition of the data we require that:

- $\bigcup_i \mathcal{C}_i = \{1, \dots, n\}$ (I.e., every data point gets assigned to a cluster.)
- $\mathcal{C}_i \cap \mathcal{C}_{i'} = \emptyset$ for $i \neq i'$ (I.e., the clusters do not overlap.)

We can now define what it means for a clustering to be "good." Specifically, our aim will be to find clusters where the data points within each cluster are tightly grouped (i.e., exhibit low variability). Recall that $\|x\|_2^2 = x^T x$, so $\|x_i - x_j\|_2$ is the euclidean distance between the vectors x_i and x_j. Mathematically, for any cluster assignment we define

$$Z(\mathcal{C}_1, \dots, \mathcal{C}_k) = \sum_{\ell=1}^{k} \frac{1}{2|\mathcal{C}_\ell|} \sum_{i,j \in \mathcal{C}_\ell} \|x_i - x_j\|_2^2 \tag{1}$$

and the smaller Z is, the better we consider the clustering. Roughly speaking, we are trying to pick a clustering that minimizes the pairwise squared distances within each cluster (normalized by the cluster sizes).

Interestingly, the definition of Z in eq. 1 can be rewritten in terms of the centroid of each cluster as

$$Z(\mathcal{C}_1, \dots, \mathcal{C}_k) = \sum_{\ell=1}^{k} \sum_{i \in \mathcal{C}_\ell} \|x_i - \mu_\ell\|_2^2, \tag{2}$$

where

$$\mu_\ell = \frac{1}{|\mathcal{C}_\ell|} \sum_{i \in \mathcal{C}_\ell} x_i$$

is the mean/centroid of the points in cluster ℓ. This formulation will be useful since it allows us to naturally think about the average of the points in a cluster as a center for the cluster. The algorithm we present later will actually use these centers to update clustering assignments.

Now that we have a formal way to define a good clustering of points, the ideal problem we would like to solve is

$$\min_{\mathcal{C}_1, \dots, \mathcal{C}_k} Z(\mathcal{C}_1, \dots, \mathcal{C}_k). \tag{3}$$

Note that we have implicitly assumed we only consider $\mathcal{C}_i$ that form valid partition and, therefore, have not explictly added those constraints. If we could efficiently solve this problem, then given k we could compute the ideal clustering of x_i into k groups (per our metric). Unfortunately, solving eq. 3 is not computationally feasible (in fact, there is a sense in which it is provably hard). Roughly speaking, we cannot do much better then simply trying all possible k way partitions of the data and seeing which one yields the smallest objective value. Nevertheless, while finding the minimizer of eq. 3 may be hard finding a good clustering is something we want to do in practice—to accomplish this we simply have to relax our goal of finding the "best" clustering.

An algorithm for k-means

The practical approach we will take finding a good clustering is to tackle eq. 3 via a local search algorithm. Given some initial clustering, we will iteratively update the cluster assignments in a manner that progressively yields a better and better clustering. However, the way in which we update the clusters will not ensure convergence to the global solution of our minimization problem. Instead, we will settle for a local solution—one where our update rule yields the same clusters we already have. The standard algorithm for solving the k-means clustering problem is known as Lloyd's algorithm[4] and is mathematically stated as:[5]

> **Input:** data $\{x_i\}_{i=1}^n$ and k
>
> Initialize $C_1, \dots, C_k$
>
> While not converged:
>
> > 1. Compute $\mu_\ell = \frac{1}{|\mathcal{C}_\ell|} \sum_{i \in \mathcal{C}_\ell} x_i$ for $\ell = 1, \dots, k$.
> > 2. Update $\mathcal{C}_1, \dots, \mathcal{C}_k$ by assigning each data point x_i to the cluster whose centroid μ_ℓ it is closest to.
> > 3. If the cluster assignments didn't change, we have converged.
>
> **Return:** cluster assignments $\mathcal{C}_1, \dots, \mathcal{C}_k$

Fig. 3 demonstrates a procession of Lloyd's algorithm (albeit somewhat inexactly since the cluster centroids are estimated). The key idea is that at each step we compute the centroid for each cluster and then reassign points to clusters based on which centroid they are closest too. This process actually has a nice feature that the objective function Z is non-increasing (and only fails to decrease if the cluster assignments have converged). Showing this fact is an interesting exercise (and may be an upcomming HW problem).

Figure 3: Lloyd's algorithm performing k-means clustering of data into three clusters.

Figure 3: Lloyd's algorithm performing k-means clustering of data into three clusters.

A more precise illustration is see in fig. 4 where data that seems to comprise of three clusters is given and the starting clustering is random. After picking an initial clustering, we find cluster centroids, reassign the points, and repeat till convergence. At each step the new cluster centroids are plotted as are the "dividing lines" that are used to assign points to clusters in the reassignment step. We will explain why these are linear later in the notes. At convergence we see that there is some ambiguity between the diffuse blue cluster and the yellow cluster near their boundary. This is partially due to the elongated ellipsoid structure of the data cluster to the right (that becomes the yellow cluster). Because we are using euclidean distances, we are making a modeling assumption that the clusters we want to find spread out roughly equally in all dimensions.

Figure 4: Lloyd's algorithm performing k-means clustering of data into three clusters.

Figure 4: Lloyd's algorithm performing k-means clustering of data into three clusters.

The cost of Lloyd's algorithm is $\mathcal{O}(ndk)$ per iteration and, therefore, in practice its efficiency is strongly dependent on how many iterations it takes to converge. While it is possible to estimate this in some special cases and develop worst case bounds, such specifics are beyond the scope of this course. It is also important to reiterate that Lloyd's algorithm does not (necessarily) solve the optimization problem in eq. 3. As we will see in the next section, poor initial cluster assignments could result in convergence to a suboptimal clustering. One reason for this is that the cluster update steps are somewhat "local," so if the ideal clustering looks very different than the current assignment the path Lloyd's algorithm takes may not lead there.

How many clusters and how they are initialized

While the k-means procedure is unsupervised, that does not mean it only takes data as input. In particular, there are two other choices the user has to make that can have a strong influence on the output. First, we have to decide how many clusters we want/expect, i.e., we have to choose k. In some settings a natural choice for k may be evident based on some underlying knowledge about the data. However, in others it may be entirely unclear. Nevertheless, the choice of k can actually be quite important in many applications.

Because we have defined a measure of cluster quality, we can use that to help choose k. In particular, let

$$Z_k \equiv Z(\mathcal{C}_1, \dots, \mathcal{C}_k)$$

be the computed objective value after we run Lloyd's algorithm for some fixed value of k. Since Z_k measures the clustering quality we might be tempted to simply compute $Z_1, Z_2, \dots$ and simply pick k based on the the smallest value of Z_k. However, this is problematic because we expect the in-cluster variation to decrease naturally with k. In fact, $Z_n = 0$ since every point is its own cluster. So, instead, we often look for the point at which Z_k stops decreasing quickly. The intuition is that as $k \to k+1$ if a single big cluster naturally subdivides into two clear clusters we will see that $Z_{k+1} \ll Z_k$. In this case we may keep increasing k to see how the objective function further evolves. In contrast, if going from $k \to k+1$ necessitates splitting up a clear cluster that does not naturally subdivide then we may see that $Z_{k+1} \approx Z_k$. An idealized version of this is illustrated in fig. 5. Once this, admittedly imprecise, criteria is met we may simply pick k as the number of clusters.

Figure 5: Choosing the number of clusters by looking for a "knee" in the objective function.

Figure 5: Choosing the number of clusters by looking for a "knee" in the objective function.

There are more rigorous ways to select k if we are willing to consider a null model for the data. For example, a common strategy is to consider the so-called gap statistic proposed by Tibshirani, Walther, and Hastie.[6] In brief, this method compares the observed cluster variability Z_k to the expected variability if the data were uniformly distributed on a rectangle that contains the data. The number of clusters k is then chosen based on the comparisons of these metrics.

The second implicit input in Lloyd's algorithm is an initial cluster assignment. This is particularly important because Lloyd's algorithm only finds a local optima. So, very different initialization could result in convergence to very different solutions. An extreme version of this is in fig. 6 where one initialization yields a good solution and one yields a solution that is far from optimal. Because of this sensitivity, a common strategy is, for each k, to try several random initial clustering assignments.[7] We then simply run Lloyd's algorithm from each initial clustering and then take the output over those runs that yielded the smallest value of $Z_k \equiv Z(\mathcal{C}_1, \dots, \mathcal{C}_k)$.

Figure 6: The result of k-means clustering for a simple data set from two different initializations—one results in a good clustering and one results in a clustering that is far from optimal.

Figure 6: The result of k-means clustering for a simple data set from two different initializations—one results in a good clustering and one results in a clustering that is far from optimal.

An alternative to thinking about initializing the cluster assignments is to randomly choose k cluster centers and then assign each point to the cluster center it is closest to (mirroring the assignment step in Lloyd's algorithm). This perspective leads to one of the most common initialization strategies for k-means known as as `k-means++`.[8] `k-means++` uses data points themselves as the initial cluster centers and is based on the idea that the cluster centers should be somewhat spread out. To accomplish this, a data point is chosen uniformly at random and selected as a center. Subsequent centers are then chosen based on a distribution weighted by the distance to the closest already selected center. In this way, it is more likely to chose a well spaced out set of k data points to form the initial cluster centers. Details of this method are available in the aforementioned reference.

Decision boundaries

As alluded to, the unsupervised learning algorithm that we choose has a big impact on the type of structure we find in the data. In the case of k-means this structure can be readily described based on how cluster centroids partition $\mathbb{R}^d$ when the cluster assignments are made. First, observe that given two cluster centroids μ_ℓ and $\mu_{\ell'}$ the set of points equidistant from the two centroids defined as $\{x \in \mathbb{R}^d \mid \|x - \mu_\ell\|_2 = \|x - \mu_{\ell'}\|_2\}$ is a line. This means that given a set of cluster centroids $\mu_i, \dots, \mu_k$ they implicitly partition space up into k domains whose boundaries are linear. Fig. 7 illustrates this point. Moreover, this shows how k-means is related to so-called Voronoi tessellations of space.

Figure 7: Cluster partition boundaries as determined by k-means clustering—they form a Voronoi tessellation of space based on the cluster centroids.

Figure 7: Cluster partition boundaries as determined by k-means clustering—they form a Voronoi tessellation of space based on the cluster centroids.

Extensions [*Optional*]

As we just observed, k-means clustering applied directly to the data $x_i \in \mathbb{R}^d$ has some important limitations. In particular, the fact that cluster boundaries are necessarily linear can be quite limiting. In fig. 8 we see a data set that seems to nicely form a few clusters, but the boundaries between those clusters is quite clearly non-linear.

Figure 8: An example where the data seems to naturally form two clusters. However, the apparent clusters cannot be separated by a linear decision boundary.

Figure 8: An example where the data seems to naturally form two clusters. However, the apparent clusters cannot be separated by a linear decision boundary.

One approach to address this problem is known as spectral clustering.[9] While the details of this approach (which has many variations) are outside the scope of this class, we provide a brief introduction to the method. The core idea behind spectral clustering is that the latent space where the data lives (for us, $\mathbb{R}^d$) is not the natural domain in which to perform k-means clustering. Therefore, we first perform a non-linear transform of the data.

Spectral clustering proceeds by first defining a similarity matrix $A \in \mathbb{R}^{n \times n}$ where $A_{i,j}$ provides some notion of similarity[10] between x_i and x_j. For example, we could use a Gaussian similarity metric $L_{i,j} = e^{-\|x_i - x_j\|_2^2/(2\sigma^2)}$ or a binary metric where $A_{i,j} = 1$ if $\|x_i - x_j\|_2 \leq \epsilon$ and 0 otherwise. Irrespective of the measure of similarity chosen, spectral clustering proceeds by forming the so-called Laplacian matrix $L = D - A$ where D is a diagonal matrix and $D_{i,i} = \sum_j A_{i,j}$. We then often let $\hat{L}$ denote the normalized Laplacian matrix $\hat{L} = I - D^{-1/2} A D^{-1/2}$.

Given $\hat{L}$ and k normalized spectral clustering[11] proceeds as follows:

1. Compute the k eigenvectors of $\hat{L}$ associated with the k algebraically smallest eigenvalues; denote the matrix of these eigenvectors $V \in \mathbb{R}^{n \times k}$.
2. Let $v_i^T = V(i,:)$ for $i = 1, \dots, n$, i.e., $v_i \in \mathbb{R}^k$ is defined as row i of the matrix V.
3. Normalize the rows via $v_i = v_i / \|v_i\|_2$ for $i = 1, \dots, n$.
4. Cluster $\{v_i\}_{i=1}^k$ into k clusters $\hat{\mathcal{C}}_1, \dots, \hat{\mathcal{C}}_k$ using k-means.
5. Place x_i into the "same" cluster as v_i was, i.e., define a clustering $\mathcal{C}_1, \dots, \mathcal{C}_k$ on x_i where $x_i \in \mathcal{C}_\ell$ if and only if $v_i \in \hat{\mathcal{C}}_\ell$.

We have still used k-means and, therefore, linear decision boundaries to partition the v_i. However, by first performing an embedding of the data via the eigenvectors of $\hat{L}$ this method allows for non-linear decision boundaries between clusters of x_i. The aforementioned references contains several examples where this proves effective.

Relation to Gaussian mixture models [*Optional*]

Another important algorithm that is related to k-means clustering is fitting Gaussian mixture models via the expectation maximization (EM) algorithm. In this setting, we assume a generative model for the data x_i and try and fit parameters of that model to the given data. This has some benefits, e.g., it allows us to assign probabilities each point is in a given cluster, and reduces to k-means in a certain parameter regime. Moreover, even if the data are not actually drawn from a Gaussian mixture model it is possible to reason about the consequences of fitting such a model to the data; though, they are beyond the scope of this course.

In particular, here we will assume that the data points x_i are drawn from a mixture of k multivariate normal distributions each with their own means μ_ℓ and covariance matrices Σ_ℓ for $\ell =, \dots, k$ and weighted by π_ℓ. To be a proper mixture model we require that the vector $\pi \in \mathbb{R}^k$ satisfies $\pi_\ell \geq 0$ for $\ell = 1, \dots, k$ and $\sum_\ell \pi_\ell = 1$. We may now formally define the density of our mixture model $\mathcal{M}(\pi, \mu_1, \dots, \mu_k, \Sigma_1, \dots, \Sigma_k)$ via

$$f(x) = \sum_\ell \pi_\ell \phi_\ell(x),$$

where ϕ_ℓ is the density of $\mathcal{N}(\mu_\ell, \Sigma_\ell)$. Another way to think about this is that to draw a data point from the mixture model $\mathcal{M}$ we first pick an integer from 1 to k based on the discrete density defined by π, say it is i, and then draw a sample from the multivariate normal distribution $\mathcal{N}(\mu_\ell, \Sigma_\ell)$.

If we knew which component of the mixture distribution (i.e., which normal distribution $\mathcal{N}(\mu_\ell, \Sigma_\ell)$$) generated each data point it would be a relatively simple problem to estimate the parameters of the individual mixture components (since we could just compute the MLE for μ_ℓ and Σ_ℓ based on the data that came from that mixture component). However, our goal is a clustering of the data—if we knew this bit of information we would be done. Instead, given x_i we would like to estimate the probability it was generated by each of the mixture components. If we knew π and all the means and covariance matrices this would also be a simple calculation. The catch is that we have none of this information and have to simultaneously estimate the membership probabilities and the parameters of the mixture model. The membership probabilities will yield our "fuzzy" clustering of each data point into each of the k groups.

To formalize this let $\Gamma \in \mathbb{R}^{n \times k}$ denote the membership probability of each data point x_i for each cluster (mixture component). Specifically, $\Gamma_{i,\ell}$ will be the membership probability for x_i into mixture component ℓ. Therefore, we require that $\Gamma_{i,\ell} \geq 0$ and $\sum_\ell \Gamma_{i,\ell} = 1$. Given the parameters of the mixture model we can explicit compute the conditional probability that x_i was drawn from mixture component ℓ as

$$\Gamma_{i,\ell} = \frac{\phi_\ell(x_i)\pi_\ell}{\sum_j \phi_j(x_i)\pi_j}.$$

We are now ready to present the EM algorithm; given data x_i as input and k it seeks to simultaneously compute membership probabilities and estimate mixture component parameters. Similar to the k-means algorithm it may only find a local optima of the underlying optimization problem (which we have omitted an explicit description of). This means that, as before, the output may be sensitive to the initialization and we are not guaranteed to find the mixture model that best describes the data. Lastly, determining convergence here is more subtle than in the case of k-means.

> **Input:** data $\{x_i\}_{i=1}^n$ and k
>
> Initialize $\hat{\pi}$, means $\hat{\mu}_1, \dots, \hat{\mu}_k$ and covariance matrices $\hat{\Sigma}_1, \dots, \hat{\Sigma}_k$
>
> While not converged:
>
> > 1. Compute group membership probabilities
> > $$\Gamma_{i,\ell} = \frac{\phi_\ell(x_i)\pi_\ell}{\sum_j \phi_j(x_i)\pi_j}.$$
> > 2. Update mixture model parameters for $\ell = 1, \dots, k$ as
> > $$\hat{\pi}_\ell = \frac{\sum_i \Gamma_{i,\ell}}{n}$$
> > $$\hat{\mu}_\ell = \frac{\sum_i \Gamma_{i,\ell} x_i}{\sum_i \Gamma_{i,\ell}}$$
> > $$\hat{\Sigma}_\ell = \frac{\sum_i \Gamma_{i,\ell}(x_i - \hat{\mu}_\ell)(x_i - \hat{\mu}_\ell)^T}{\sum_i \Gamma_{i,\ell}}$$
> > 3. Check for convergence
>
> **Return:** Group membership probabilities Γ and mixture model parameter estimates $\hat{\pi}, \hat{\mu}_1, \dots, \hat{\mu}_k, \hat{\Sigma}_1, \dots, \hat{\Sigma}_k$

Typically, the two core steps of this algorithm are referred to as expectation steps and maximization steps. Step 1 is the expectation step where we compute conditional probabilities that represent group membership. Step 2 is then the maximization step where given a set of class membership probabilities we compute maximum likelihood estimates for the mixture model components. As before, there are several strategies to try and compute a good initialization for the necessary parameters.

If we return to the k-means algorithm it has a similar flavor. Assigning points based on the cluster centroids is analogous to the expectation step and updating the centroids is analogous to the maximization step. The connection can actually be made more explicit. In fact, we we restrict the covariance matrics to be $\Sigma_\ell = \sigma^2 I$ and let $\sigma \to 0$ the EM algorithm for a Gaussian mixture model essentially becomes the k-means algorithm.

---

1. We will also briefly discuss how similar methods can be extended to incorporate "similarities" between features that are defined more generally than by ℓ_2 distance.↩
2. For the moment, assume k is given; we will return to this point later.↩
3. Much of this section could be written with a more general distance metric (e.g., $\|\cdot\|_1$, $\|\cdot\|_\infty$, or any well-defined metric $d(\cdot,\cdot)$). In fact, if we use $\|\cdot\|_1$ we recover an algorithm known as k-medians. However, in the general case it may not always be so easy to write down the appropriate update formula for the cluster assignments.↩
4. While Lloyd proposed the algorithm in 1957, an associated publication of his work did not appear till 1982 as Lloyd, S. P. "Least squares quantization in PCM" *IEEE Transactions on Information Theory* 28 (2), 129–137, (1982).↩
5. Some aspects of this algorithm may seem under specified—we will get to them later.↩
6. See Tibshirani R., Walther G., and Hastie T. "Estimating the number of clusters in a data set via the gap statistic" *J. R. Statist. Soc. B* 63 (2), 411-423, (2001).↩
7. There are many ways to assign an initial clustering "at random." For example, for each data point we could independently assign it to a cluster uniformly at random (i.e., point x_i ends up in cluster ℓ w.p. $1/k$).↩
8. See Arthur, D. and Vassilvitskii, S. "k-means++: the advantages of careful seeding" *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 1027–1035, (2007).↩
9. One good resource to learn more is Von Luxburg, U. "A tutorial on spectral clustering." *Statistics and computing* 17 (4), 395-416, 2007. A version is available on arXiv.↩
10. We do require that $A_{i,j} \geq 0$ and $A_{i,j} = A_{j,1}$.↩
11. See Ng, A.Y., Jordan, M.I. and Weiss, Y., "On spectral clustering: Analysis and an algorithm,"" *In Advances in neural information processing systems* 849-856, 2002.↩