Mathematical Introduction to Machine Learning

Lecture 4: Unsupervised Learning I

November 17, 2024

Lecturer: Lei Wu *Scribe: Lei Wu*

A basic requirement for supervised learning is that the data must be labeled. In many applications, labeling the data is significantly more expensive than obtaining raw unlabeled data. For example labeling medical images requires well-trained physicians. This motivates us to develop models that can be used to exploit unlabeled data. This is the task of unsupervised learning.

In unsupervised learning, we are given a set of unlabeled data $S = \{\mathbf{x}_i\}_{i=1}^n$ with $\mathbf{x}_i$ drawn from an underlying distribution μ^*, and the task is to discover some *useful information* about μ^*. The definition of useful information may vary with the applications. In this chapter, we will cover the following three categories of unsupervised learning tasks.

- **Density estimation:** Approximating the underlying probability distribution μ^*.
- **Dimension reduction:** The data may approximately concentrate on a set whose dimensionality is much lower than that of the original space. The task of dimension reduction is to identify the effective description of this low-dimensional structure.
- **Clustering:** If μ^* is multi-modal, S can be decomposed into several clusters. Finding these clusters is an important problem in many applications.

Among these tasks, density estimation is the most ambitious and the most challenging task since it aims to learn the whole data distribution. By contrast, the objective of dimension reduction and clustering is limited to finding some aspects of the distribution.

1 Density estimation

Let us start with density estimation. The original density estimation task is limited to the setting when μ^* is absolutely continuous with respect to the Lebesgue measure with density ρ^* and the task is to approximate ρ^* using the samples $\{\mathbf{x}_i\}_{i=1}^n$. We will take a more general viewpoint of trying to approximate μ^* regardless whether it has a density or not.

An obvious candidate is given by the empirical distribution:

$$\hat{\mu}_n(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^n \delta(\mathbf{x} - \mathbf{x}_i). \tag{1}$$

However, we are not satisfied with $\hat{\mu}_n$, since it generalizes badly on unseen samples in the sense that it cannot provide meaningful prediction of their probability. Moreover, it is impossible to generate new samples from $\hat{\mu}_n$. Mathematically speaking, $\hat{\mu}_n$ is too singular and we want an approximation that is smoother such that we can extrapolate to unseen data. This is typically achieved by adding regularization.

It is often the case that μ^* is singular with respect to the Lebesgue measure and the singularity is usually quite anisotropic. Consequently, adding the right regularizations is a highly nontrivial issue. Nevertheless, we will introduce two types of density estimators, which work reasonably well when the input dimension d is relatively small.

1.1 The histogram estimator

For simplicity, assume that the support of μ^* is $I_d := [0,1]^d$. Divide I_d into the union of small cubes of equal size h,

$$I_d = \cup_j B_j.$$

The volume of each cube is h^d. Let n_j be the number of samples in S that fall into B_j. Then, $\sum_j n_j = n$. The histogram estimator is given by

$$\hat{\rho}_h(\mathbf{x}) = \frac{1}{nh^d} \sum_j n_j \mathbf{1}_{B_j}(\mathbf{x}), \tag{2}$$

where $\mathbf{1}_{B_j}$ is the indicator function of B_j

$$\mathbf{1}_{B_j}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in B_j \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Obviously, the histogram estimator is piecewise constant.

Consider the example $\mu^* = 0.5\mathcal{N}(0,1) + 0.5\mathcal{N}(3,1)$ and we randomly draw $n = 500$ samples from μ^*. Figure 1 shows the corresponding histogram estimator with $h = 0.05$. As can be seen, this piecewise constant estimator is not ideal since μ^* or ρ^* is very smooth. This motivates us to consider improved smoothness in the construction of density estimators. One possible approach is to replace the hard box (3) with certain smoothed "probabilistic box". One way of implementing this idea is to use the kernel density estimator (KDE), a generalization of the histogram estimator.

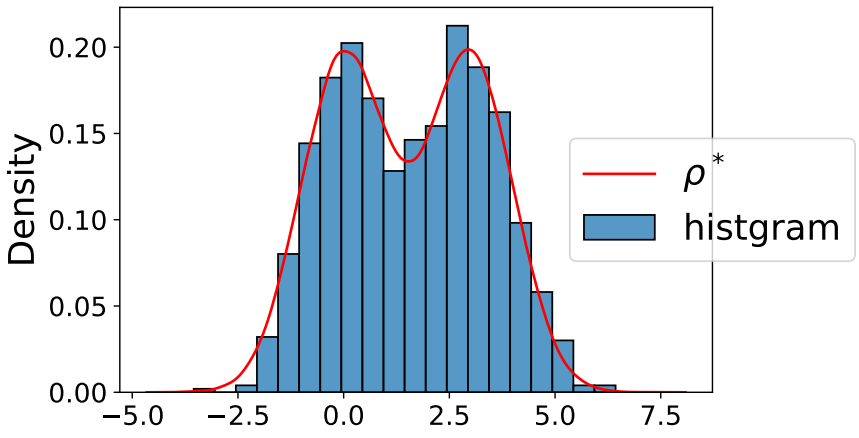

Figure 1: An illustration of the histogram estimator.

1.2 Kernel density estimator (KDE)

In KDE, we replace the delta function in (1) by a "smooth version". To this end, we introduce the the smoothing kernel function $K : \mathbb{R}^d \mapsto \mathbb{R}$. K should satisfy the following properties:

- $K(\mathbf{x}) \geq 0$ for all $\mathbf{x}$,

- $\int K(\mathbf{x})d\mathbf{x} = 1$,
- $K(\mathbf{x}) = K(-\mathbf{x})$.

Let $K_h(\mathbf{x}) = h^{-d}K(\mathbf{x}/h)$. Then, $K_h(\cdot) \to \delta(\cdot)$ as $h \to 0$ in the sense that for any bounded continuous function f

$$\begin{aligned}\lim_{h\to 0}\int (K_h(\mathbf{x}) - \delta(\mathbf{x}))f(\mathbf{x})d\mathbf{x} &= \lim_{h\to 0}\int K(\mathbf{z})(f(h\mathbf{z}) - f(0))d\mathbf{z} \\ &= \int K(\mathbf{z})\lim_{h\to 0}(f(h\mathbf{z}) - f(0))d\mathbf{z} = 0 \end{aligned} \tag{4}$$

The following is a list of popular smoothing kernels for $d = 1$.

- Box kernel: $K(x) = 1_{[-1/2,1/2]}$.
- Triangular kernel: $K(x) = \max(0, 1 - |x|)$.
- Gaussian kernel: $K(x) = \frac{1}{(2\pi)^{1/2}}e^{-\frac{x^2}{2}}$

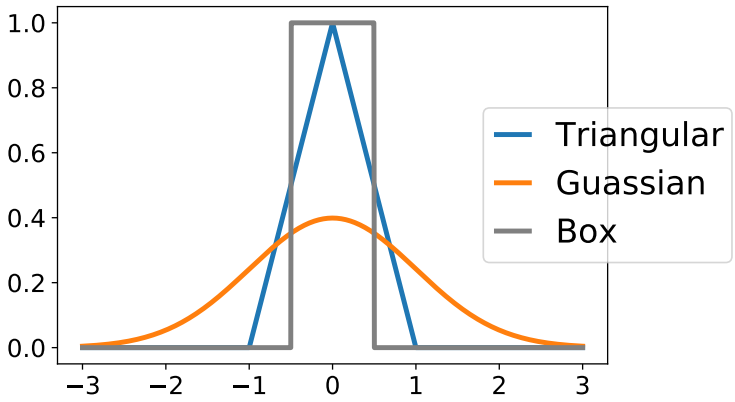

Figure 2: Various smoothing kernels for $d = 1$.

Figure 2 provides a visualization of these smoothing kernels. An obvious difference is their smoothness. Box kernel is piecewise constant; triangular kernel is piecewise linear; Gaussian kernel is infinitely differentiable in the whole space. When $d > 1$, one can construct the smoothing kernel using one-dimensional ones:

$$K(\mathbf{x}) = \prod_{j=1}^{d} K(x_j). \tag{5}$$

We remark that the smoothing kernel constructed in (5) is isotropic. One can also construct anisotropic smoothing kernels.

The kernel density estimator (KDE) is given by

$$\hat{\rho}_h(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} K_h(\mathbf{x} - \mathbf{x}_i) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h^d}K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right). \tag{6}$$

Using the properties of the smoothing kernel, it is easy to show that $\hat{\rho}_h$ satisfies:

- $\hat{\rho}_h(\mathbf{x}) \geq 0$ for any $\mathbf{x} \in \mathbb{R}^d$.
- $\int \hat{\rho}_h(\mathbf{x})\,\mathrm{d}\mathbf{x} = \frac{1}{n}\sum_{i=1}^n \int \frac{1}{h^d} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)\mathrm{d}\mathbf{x} = \frac{1}{n}\sum_{i=1}^n \int K(\mathbf{x}')\,\mathrm{d}\mathbf{x}' = 1.$

Therefore, $\hat{\rho}_h$ is indeed a probability density function.

If we take the box kernel, KDE becomes an "adaptive" histogram estimator in the sense that the box positions are adapted to the data. Note that the box positions in the histogram estimator are specified by the uniform grids and independent of the data.

In KDE, there is an important hyper-parameter: the *bandwidth* h, which controls the smoothness of the "smoothed delta function". Obviously,

$$\hat{\rho}_h = \frac{1}{n}\sum_{i=1}^n K_h(\cdot - \mathbf{x}_i) \to \frac{1}{n}\sum_{i=1}^n \delta(\cdot - \mathbf{x}_i) \quad \text{as } h \to 0. \tag{7}$$

When h is small, $\hat{\rho}_h$ is close to the empirical distribution. When h is large, $\hat{\rho}_h$ is nearly flat. Choosing the right value of h is a key problem in KDE.

To see how the performance of KDE is affected by the bandwidth, we plot the KDE estimators for varying bandwidths in Figure 3. Here, we randomly sample $n = 500$ points from $\mu^* = 0.5\mathcal{N}(0,1) + 0.5\mathcal{N}(3,1)$. One can see that

- When h is too small, the estimator is extremely rugged.
- When h is too large, the estimator is too flat. In this case, it fails to identify the two modes of ρ^*.
- When $h = 0.12$, KDE recovers μ^* pretty well.

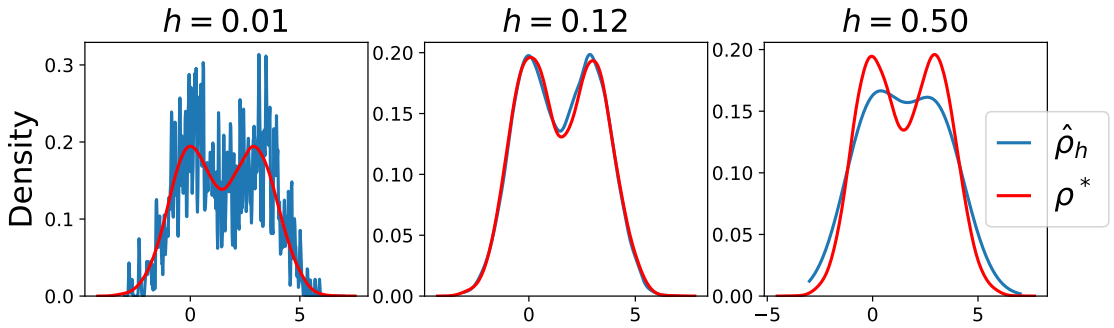

Figure 3: KDE with the Gaussian kernel with varying bandwidths. Here $n = 500$. The red curve corresponds to the ground truth.

1.2.1 Error analysis

Let $\rho_h(\mathbf{x}) = \mathbb{E}_S[\hat{\rho}_h(\mathbf{x})]$, where $\mathbb{E}_S$ means taking expectation with respect to the sampling of $S = \{\mathbf{x}_i\}_{i=1}^n$. Then,

$$\rho_h(\mathbf{x}) = \mathbb{E}_S\left[\frac{1}{n}\sum_{i=1}^n K_h\left(\mathbf{x} - \mathbf{x}_i\right)\right] = \frac{1}{n}\sum_{i=1}^n \mathbb{E}_{\mathbf{x}_i}\left[K_h\left(\mathbf{x} - \mathbf{x}_i\right)\right]$$

$$= \mathbb{E}_{\mathbf{x}' \sim \rho^*}[K_h(\mathbf{x} - \mathbf{x}')] = \int K_h(\mathbf{x} - \mathbf{x}')\rho^*(\mathbf{x}')\,\mathrm{d}\mathbf{x}'. \tag{8}$$

The error at any $\mathbf{x} \in \mathbb{R}^d$ can be decomposed as follows.

$$\begin{aligned}
\mathbb{E}_S\,|\hat{\rho}_h(\mathbf{x}) - \rho^*(\mathbf{x})|^2 &= \mathbb{E}_S\,|\hat{\rho}_h(\mathbf{x}) - \rho_h(\mathbf{x})|^2 + \mathbb{E}_S\,|\rho_h(\mathbf{x}) - \rho^*(\mathbf{x})|^2 \\
&\quad + 2\,\mathbb{E}_S[(\hat{\rho}_h(\mathbf{x}) - \rho_h(\mathbf{x}))(\rho_h(\mathbf{x}) - \rho^*(\mathbf{x}))] \\
&= \underbrace{\mathbb{E}_S\,|\hat{\rho}_h(\mathbf{x}) - \rho_h(\mathbf{x})|^2}_{variance} + \underbrace{|\rho_h(\mathbf{x}) - \rho^*(\mathbf{x})|^2}_{bias},
\end{aligned} \tag{9}$$

where in the second equality we used $\rho_h(\mathbf{x}) = \mathbb{E}_S[\hat{\rho}_h(\mathbf{x})]$. This is the standard bias-variance decomposition, and we will estimate the two terms separately.

First we look at the bias term.

$$\begin{aligned}
\rho_h(\mathbf{x}) - \rho^*(\mathbf{x}) &= \int \frac{1}{h^d} K\left(\frac{\mathbf{x} - \mathbf{x}'}{h}\right) \rho^*(\mathbf{x}')\,\mathrm{d}\mathbf{x}' - \rho^*(\mathbf{x}) \quad \text{(use Eq.(8))} \\
&= \int K(\mathbf{z})\rho^*(\mathbf{x} + h\mathbf{z})\,\mathrm{d}\mathbf{z} - \rho^*(\mathbf{x}) \\
&= \int K(\mathbf{z})(\rho^*(\mathbf{x} + h\mathbf{z}) - \rho^*(\mathbf{x}))\,\mathrm{d}\mathbf{z} \quad \left(\text{use } \int K(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1\right) \\
&= \int K(\mathbf{z})\left(h\langle \nabla\rho^*(\mathbf{x}), \mathbf{z}\rangle + h^2\langle \mathbf{z}, \nabla^2\rho^*(\mathbf{x})\mathbf{z}\rangle + o(h^2)\right)\mathrm{d}\mathbf{z} \\
&= 0 + h^2 \int K(\mathbf{z})\mathbf{z}^T\nabla^2\rho^*(\mathbf{x})\mathbf{z}\,\mathrm{d}\mathbf{z} + o(h^2),
\end{aligned}$$

where in the last equality we used the fact that $\int \mathbf{z}K(\mathbf{z})\,\mathrm{d}\mathbf{z} = 0$. Assume that $C_2 = \max_{\mathbf{x}} \|\nabla^2\rho^*(\mathbf{x})\|_2 < \infty, C_1 = \int K(\mathbf{z})\|\mathbf{z}\|^2\,\mathrm{d}\mathbf{z} < \infty$. Then, we have

$$|\rho_h(\mathbf{x}) - \rho^*(\mathbf{x})| \le C_1C_2h^2. \tag{10}$$

Next we consider the variance term. Let $Z_i = K_h(\mathbf{x} - \mathbf{x}_i)$. Then, $\{Z_i\}$ are iid random variables with $\mathbb{E}_{\mathbf{x}_i}[Z_i] = \rho_h(\mathbf{x})$. According to Eq. (8),

$$\begin{aligned}
\mathbb{E}_S\,|\hat{\rho}_h(\mathbf{x}) - \rho_h(\mathbf{x})|^2 &= \mathbb{E}_S\,\Big|\frac{1}{n}\sum_i (Z_i - \mathbb{E}[Z_i])\Big|^2 \\
&= \mathrm{Var}\left(\frac{1}{n}\sum_i Z_i\right) = \frac{1}{n}\mathrm{Var}(Z_1) \\
&\le \frac{1}{n}\,\mathbb{E}[Z_1^2] = \frac{1}{nh^{2d}} \int K^2\left(\frac{\mathbf{x} - \mathbf{x}'}{h}\right)\rho^*(\mathbf{x}')\,\mathrm{d}\mathbf{x}' \\
&= \frac{1}{nh^d} \int K^2(\mathbf{z})\rho^*(\mathbf{x} + h\mathbf{z})\,\mathrm{d}\mathbf{z}
\end{aligned} \tag{11}$$

Assume that $\sup_{\mathbf{x}} \rho^*(\mathbf{x}) \le C, \int K^2(\mathbf{z})\,\mathrm{d}\mathbf{z} \le C$. Then,

$$\mathbb{E}_S\,|\hat{\rho}_h(\mathbf{x}) - \rho_h(\mathbf{x})|^2 \le \frac{C}{nh^d}. \tag{12}$$

Combining (10) and (12), we have the following error estimate of KDE.

Theorem 1.1. *Under the previous assumptions on ρ^* and K, the mean squared error of $\hat{\rho}_h$ satisfies*

$$\mathrm{MSE}(\hat{\rho}_h) := \mathbb{E}_S \int |\hat{\rho}_h(\mathbf{x}) - \rho^*(\mathbf{x})|^2 \, \mathrm{d}\mathbf{x} \leq C \left(h^4 + \frac{1}{nh^d} \right). \tag{13}$$

Consequently, the optimal choice of h satisfies $h_{opt} = Cn^{-\frac{1}{4+d}}$ and the corresponding optimal error rate is bounded by

$$\mathrm{MSE}(\hat{\rho}_{h_{opt}}) \leq C \left(\frac{1}{n} \right)^{\frac{4}{4+d}}. \tag{14}$$

This error estimate reveals the fact that the KDE suffers from the curse of dimensionality.

1.2.2 Cross-validation

To find a more practical way of choosing h, we introduce a data-based approach to estimate the error

$$\begin{aligned} \int (\hat{\rho}_h(\mathbf{x}) - \rho^*(\mathbf{x}))^2 \, \mathrm{d}\mathbf{x} &= \int \hat{\rho}_h(\mathbf{x})^2 \, \mathrm{d}\mathbf{x} - 2 \int \hat{\rho}_h(\mathbf{x}) \rho^*(\mathbf{x}) \, \mathrm{d}\mathbf{x} + \int \rho^*(\mathbf{x})^2 \, \mathrm{d}\mathbf{x} \\ &= \int \hat{\rho}_h(\mathbf{x})^2 \, \mathrm{d}\mathbf{x} - 2 \, \mathbb{E}_{\mathbf{x} \sim \rho^*}[\hat{\rho}_h(\mathbf{x})] + \int \rho^*(\mathbf{x})^2 \, \mathrm{d}\mathbf{x}. \end{aligned} \tag{15}$$

The last term is a constant. Hence, we only need to estimate

$$R(h) := \int \hat{\rho}_h(\mathbf{x})^2 \, \mathrm{d}\mathbf{x} - 2 \, \mathbb{E}_{\mathbf{x} \sim \rho^*}[\hat{\rho}_h(\mathbf{x})]. \tag{16}$$

The first term can be computed directly. The second term can be estimated by Monte-Carlo method if we have in our hands a new dataset S'

$$\hat{R}_{S'}(h) := \int \hat{\rho}_h(\mathbf{x})^2 \, \mathrm{d}\mathbf{x} - \frac{2}{|S'|} \sum_{\mathbf{x} \in S'} \hat{\rho}_h(\mathbf{x}). \tag{17}$$

Note that $\hat{\rho}_h$ is learned from the training set S and S' are a new dataset. Thus S' are independent of $\hat{\rho}_h$. This independence is important to gurantee the smallness of the Monte-Carlo approximation:

$$\frac{1}{|S'|} \sum_{\mathbf{x} \in S'} \hat{\rho}_h(\mathbf{x}) - \mathbb{E}_{\mathbf{x} \sim \rho^*}[\hat{\rho}_h(\mathbf{x})] \sim \frac{\sqrt{\mathbf{Var}_{x \sim \rho^*}[\hat{\rho}_n^2(x)]}}{|S'|}.$$

This motivates a simple validation method. Randomly split the data into two disjoint sets: $S = S_1 \cup S_2$. Let $n_i = |S_i|$. Then $n_1 + n_2 = n$. The simple validation method for selecting h goes as follows.

- Calculate $\hat{\rho}_h$ using the set S_1.
- Estimate $R(h)$ using S_2 and Eq. (17).
- $\hat{h}_{opt} = \mathrm{argmin}_h \, \hat{R}_{S_2}(h)$.

S_1, S_2 is called the training set and *validation set*, respectively. $\hat{R}_{S_2}(h)$ is usually called the validation error, which is an estimate of the true error $R(h)$ using the validation data.

The cross-validation method is a generalization of the proceeding procedure. A k-fold cross-validation method splits the data into k disjoint subsets: $S = \cup_{j=1}^{k} S_j$. Denote by $\hat{\rho}_h^{(-j)}$ the KDE constructed from $S \setminus S_j$. Then, we estimate the error of $\hat{\rho}_h^{(-j)}$ using S_j. Repeat this procedure for $j = 1, \ldots, k$. The average of the errors is the cross-validation error:

$$\hat{R}_{\text{cv}}(h) = \frac{1}{k} \sum_{j=1}^{n} \left(\int \hat{\rho}_h^{(-j)}(\mathbf{x})^2 \, \mathrm{d}\mathbf{x} - \frac{2}{|S_j|} \sum_{\mathbf{x} \in S_j} \rho_h^{(-j)}(\mathbf{x}) \right). \tag{18}$$

Then, return $\hat{h}_{opt} = \text{argmin}_h \hat{R}_{\text{cv}}(h)$.

2 Dimension Reduction

In this section, we assume that the $\mathbf{x}_i$'s lie (approximately) on a low-dimensional subset. See Figure 4 for some illustration. The task of dimension reduction is to identify the low-dimensional structure. Mathematically speaking, our objective is to look for a mapping $f : \mathbb{R}^d \to \mathbb{R}^k$ with $k \ll d$, such that $\{f(\mathbf{x}_i)\}_{i=1}^n$ "preserves almost all the information" contained in $\{\mathbf{x}_i\}_{i=1}^d$. Alternatively, we can also look for a mapping $g : \mathbb{R}^k \to \mathbb{R}^d$ with $k \ll d$ such that the average distance between $g(\mathbf{z}_i)$ and $\mathbf{x}_i (i \in [n])$ is minimized.

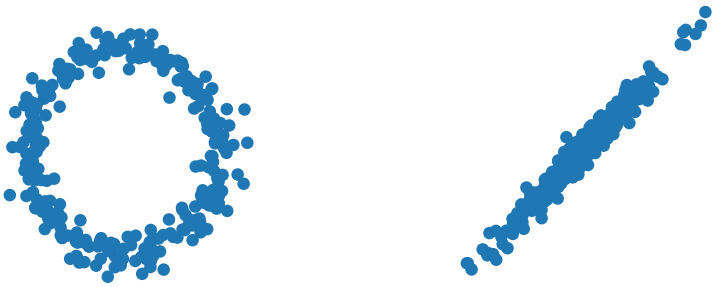

Figure 4: Illustration of the datasets that lie effectively on some low-dimensional subsets.

2.1 Principle component analysis (PCA)

PCA is by far the most popular linear dimension reduction method. Denote by $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)^T \in \mathbb{R}^{n \times d}$ the data matrix. In PCA we look for a k-dimensional *affine subspace* that best represents the n data points. See Figure 5 for an illustration.

Define g by $g(\mathbf{z}) = W\mathbf{z} + \mathbf{c}$ with $W \in \mathbb{R}^{d \times k}, \mathbf{z} \in \mathbb{R}^k, \mathbf{c} \in \mathbb{R}^d$. The objective of PCA is to minimize

$$\min_{W, \mathbf{c}, \mathbf{z}_1, \ldots, \mathbf{z}_n} L(W, \mathbf{c}, Z) = \sum_{i=1}^{n} \|\mathbf{x}_i - (W\mathbf{z}_i + \mathbf{c})\|_2^2. \tag{19}$$

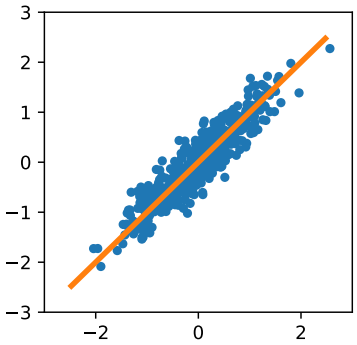

Figure 5: An illustration of PCA.

Notice that for any invertible matrix $Q \in \mathbb{R}^{k\times k}$, the transformation $(W, \{\mathbf{z}_i\}) \to (WQ, \{Q^{-1}\mathbf{z}_i\})$ leaves the value of the objective function L invariant. Hence, we can assume without loss of generality that $W^TW = I_k$. We also assume that $\sum_{i=1}^n \mathbf{z}_i = 0$, since we can always absorb it into $\mathbf{c}$.

First we take W to be fixed, and compute

$$\frac{\partial L}{\partial \mathbf{c}} = 2\sum_{i=1}^n (\mathbf{x}_i - W\mathbf{z}_i - c) = 0 \quad \Rightarrow \quad \mathbf{c} = \bar{\mathbf{x}} := \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i$$
$$\frac{\partial L}{\partial \mathbf{z}_i} = 2W^T(\mathbf{x}_i - W\mathbf{z}_i - c) = 0 \quad \Rightarrow \quad \mathbf{z}_i = W^T(\mathbf{x}_i - \mathbf{c})$$

Substituting them into (19), the objective becomes

$$\begin{aligned}\sum_{i=1}^n \|\mathbf{x}_i - (W\mathbf{z}_i + \mathbf{c})\|_2^2 &= \sum_{i=1}^n \|\mathbf{x}_i - (WW^T(\mathbf{x}_i - \mathbf{c}) + \mathbf{c})\|_2^2 \\ &= \sum_{i=1}^n \|(I - WW^T)(\mathbf{x}_i - \bar{\mathbf{x}})\|_2^2 \quad (\mathbf{c} = \bar{\mathbf{x}}) \\ &= \sum_{i=1}^n (\|(\mathbf{x}_i - \bar{\mathbf{x}})\|_2^2 - \|W^T(\mathbf{x}_i - \bar{\mathbf{x}})\|_2^2)\end{aligned}$$

Note that $\|W^T(\mathbf{x}_i - \bar{\mathbf{x}})\|_2^2 = \|WW^T(\mathbf{x}_i - \bar{\mathbf{x}})\|_2^2$, where the later is the orthogonal projection of $\mathbf{x}_i - \bar{\mathbf{x}}$ to the subspace spanned by $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k\}$. So PCA can also be formulated as being looking for a subspace such that the total error after the projection is minimized. Figure 6 provides an illustration.

Therefore, PCA is equivalent to maximizing

$$\sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}}_i)^T WW^T(\mathbf{x}_i - \bar{\mathbf{x}}_i) = \mathrm{Tr}(WW^T \sum_i (\mathbf{x}_i - \bar{\mathbf{x}}_i)(\mathbf{x}_i - \bar{\mathbf{x}}_i)^T).$$

Let $\hat{\Sigma}_n = \frac{1}{n}\sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T)$ be the empirical covariance matrix. The original problem is equivalent to

$$\max_{W^TW = I_k} \mathrm{tr}(WW^T\hat{\Sigma}_n) = \max_{W^TW = I_k} \mathrm{tr}(W^T\hat{\Sigma}_n W). \tag{20}$$

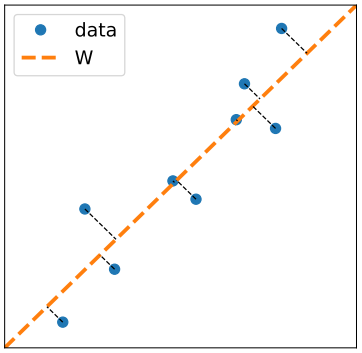

Figure 6: PCA: minimizing the total residual of the projection.

Let $\hat{\Sigma}_n = \sum_{i=1}^{d} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$ ($\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$) be the eigen-decomposition of $\hat{\Sigma}_n$. From the Rayleigh's variational principle, we conclude that $\mathbf{w}_j = \mathbf{u}_j$ for $j = 1, \ldots, k$ and

$$\max_{W^T W = I_k} \text{tr}(W^T \hat{\Sigma}_n W) = \sum_{i=1}^{k} \lambda_k. \tag{21}$$

In words, the solution of PCA is found in the top k eigenvectors of the empirical covariance matrix, which are also called the *principal components*.

PCA as a low-rank approximation. Without loss of generality, assume that $\bar{\mathbf{x}} = 0$ and rewrite the original objective function as

$$\min_{W^T W = I_k, \mathbf{z}_1, \ldots, \mathbf{z}_n} \sum_{i=1}^{n} \|\mathbf{x}_i - W\mathbf{z}_i\|_2^2 = \min_{W^T W = I_k, Z \in \mathbb{R}^{k \times n}} \|X - WZ\|_F^2. \tag{22}$$

Lemma 2.1. *The formulation* (22) *is equivalent to*

$$\min_{rank(Y) \leq k, Y \in \mathbb{R}^{n \times d}} \|X - Y\|_F^2. \tag{23}$$

Proof. Denote by Y^* the solution of (23). Consider the SVD decomposition $Y^* = U\Sigma V$ with $U \in \mathbb{R}^{n \times k}, V^T \in \mathbb{R}^{k \times k}$. Then, $W = U, Z = \Sigma V^T$ is a solution of (22). Let (W, Z) be a solution of (22). Then, $\hat{X} = WZ$ must be a solution of (23), otherwise we can construct a better solution for (22) using the SVD decomposition. □

Notice that (23) can be written equivalently as

$$\begin{aligned} \min \quad & \|E\|_F^2 \\ s.t. \quad & X = Y + E \\ & \text{rank}(Y) \leq k. \end{aligned} \tag{24}$$

We can view this as a decomposition of X into the sum of a low-rank matrix and a small error matrix (in the sense of Frobenius norm).

Robust PCA. The previous interpretation naturally lead to an extension: robust PCA [Candès et al., 2011]. The idea is to look for the following decomposition:

$$X = Y + E, \tag{25}$$

where Y is low-rank and E is sparse. Figure 7 shows an application of robust PCA in video analysis.

The robust PCA problem is non-convex since both the sparsity and rank constraint are non-convex. In practice, one usually considers a convex relaxation. As we have seen in Lasso, we can replace the ℓ_0 norm (i.e. the number of non-zero entries) used to measure sparsity by the ℓ_1 norm. For the rank function, one can use the following relaxation:

$$rank(L) = \#\{i : \sigma_i(L) \neq 0\} \Rightarrow \|L\|_* = \sum_i \sigma_i(L), \tag{26}$$

where $\|L\|_*$ is called the *nuclear norm* of L. Therefore, the objective function of the relaxed problem becomes

$$\min_{Y+E=X} \|Y\|_* + \lambda \|E\|_1, \tag{27}$$

where $\|E\|_1 = \sum_{i,j} |E_{i,j}|$. This problem can be solved using standard convex optimization algorithms [Boyd et al., 2004].

Figure 7: Surveillance video as low rank plus sparse matrices: Left = low rank (middle) + sparse (right). The background image leads to a rank-1 component and the occasional appearance of the customers contributes to the sparse component .

2.2 Kernel PCA

The basic assumption of PCA is that the data concentrate on a low-dimensional linear subspace. This assumption is quite restrictive for many applications, where the low-dimensional structure can be nonlinear (see, e.g., the left of Figure 8 for an illustration). In this section, we introduce kernel PCA, a convenient way of extending PCA to the nonlinear setting. It can be derived using the kernel trick we introduced previously **??**.

Let $k : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ be a kernel and the corresponding feature map is given by $\Phi : \mathbb{R}^d \mapsto \mathbb{R}^D$. Then,

$$k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle.$$

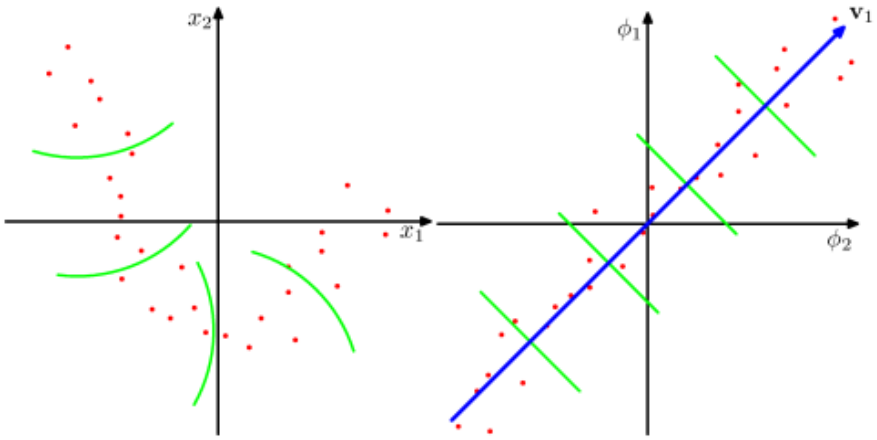

Figure 8: Semantic illustration of kernel PCA. Data points on the left lie approximately on a 1-dimensional curve. The green line indicates the orthogonal projection. (Taken from *Pattern Recognition and Machine Learning* by Christopher M. Bishop)

Here, we assume the feature space is $\mathbb{R}^D$ instead of a general Hilbert space for simplicity. One can think of D as being infinite.

We first map the data into the feature space:

$$\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \mapsto \{\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n)\}.$$

Assume that the data in the feature space concentrate on a low-dimensional linear subspace. We can apply standard PCA in the feature space. Assume as before that $\frac{1}{n}\sum_{i=1}^n \Phi(\mathbf{x}_i) = 0$. Let $\Sigma = \frac{1}{n}\sum_{i=1}^n \Phi(\mathbf{x}_i)\Phi(\mathbf{x}_i)^T \in \mathbb{R}^{D\times D}$ be the feature covariance matrix. Our objective becomes

$$\max_{W^TW=I_k} W^T\Sigma W. \tag{28}$$

The solution is given by the eigenvectors of Σ:

$$\Sigma\mathbf{v} = \lambda\mathbf{v}. \tag{29}$$

Note that a vector $\mathbf{w} \in \mathbb{R}^D$ corresponds to a nonlinear function in the original space:

$$p_{\mathbf{w}}(x) = \langle \mathbf{w}, \Phi(\mathbf{x})\rangle.$$

Theorem 2.2 (Representer theorem)**.** *Let $(\lambda, \mathbf{v})$ be the eigenpair of Σ. If $\lambda \neq 0$, $\mathbf{v}$ can be expressed as a linear combination of the features:*

$$\mathbf{v} = \sum_{i=1}^n \alpha_i \Phi(\mathbf{x}_i).$$

Proof. By definition,

$$\mathbf{v} = \frac{1}{\lambda}\Sigma\mathbf{v} = \frac{1}{\lambda}\sum_{i=1}^n \Phi(\mathbf{x}_i)\Phi(\mathbf{x}_i)^T\mathbf{v} = \sum_{i=1}^n \alpha_i\Phi(\mathbf{x}_i).$$

□

The above theorem suggests that learning $\mathbf{v} \in \mathbb{R}^D$ is equivalent to looking for $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^T \in \mathbb{R}^n$. This reduces the original D-dimensional problem to a n-dimensional problem. The principal component corresponding to $\mathbf{v}$ is given by

$$p_{\mathbf{v}}(\mathbf{x}) = \langle \mathbf{v}, \Phi(\mathbf{x}) \rangle = \langle \sum_{i=1}^{n} \alpha_i \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \rangle = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}). \tag{30}$$

Obviously they are nonlinear functions.

Derivation of the kernel PCA. For simplicity, we focus on the case when $k = 1$, i.e., $W = \mathbf{v} \in \mathbb{R}^D$. Plugging $\mathbf{v} = \sum_{j=1}^{n} \alpha_j \Phi(\mathbf{x}_j)$ into Eq. (29), we have

$$\sum_{i=1}^{n} \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_i)^T (\sum_{j=1}^{n} \alpha_j \Phi(\mathbf{x}_j)) = \lambda \sum_{j=1}^{n} \alpha_j \Phi(\mathbf{x}_j),$$

which leads to

$$\sum_{i=1}^{n} \Phi(\mathbf{x}_i) \sum_{j=1}^{n} k(\mathbf{x}_i, \mathbf{x}_j) \alpha_j = \lambda \sum_{j=1}^{n} \alpha_j \Phi(\mathbf{x}_j).$$

Multiplying this with $\Phi(\mathbf{x}_s)$, we have

$$\sum_{i=1}^{n} k(\mathbf{x}_s, \mathbf{x}_i) \sum_{j=1}^{n} k(\mathbf{x}_i, \mathbf{x}_j) \alpha_j = \lambda \sum_{j=1}^{n} \alpha_j k(\mathbf{x}_s, \mathbf{x}_j),$$

which leads to

$$K\boldsymbol{\alpha} = \lambda \boldsymbol{\alpha}.$$

Here, $K = (k(\mathbf{x}_i, \mathbf{x}_j)) \in \mathbb{R}^{n \times n}$ is the kernel matrix.

The normalization goes as follows

$$\|\mathbf{v}\|^2 = 1 \Rightarrow \sum_{i,j=1}^{m} \alpha_i \alpha_j \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j) = \boldsymbol{\alpha}^T K \boldsymbol{\alpha} = 1 \tag{31}$$

Using the fact that $K\boldsymbol{\alpha} = \lambda \boldsymbol{\alpha}$, we have

$$\boldsymbol{\alpha}^T K \boldsymbol{\alpha} = \lambda \|\boldsymbol{\alpha}\|^2 = 1 \Rightarrow \|\boldsymbol{\alpha}\|^2 = \frac{1}{\lambda}. \tag{32}$$

In practice, we need to center the features by introducing:

$$\tilde{\Phi}(\mathbf{x}_i) = \Phi(\mathbf{x}_i) - \frac{1}{n} \sum_{s=1}^{n} \Phi(\mathbf{x}_s).$$

The corresponding kernel becomes:

$$\tilde{k}(\mathbf{x}_i, \mathbf{x}_j) = \left(\Phi(\mathbf{x}_i) - \frac{1}{n} \sum_{s=1}^{n} \Phi(\mathbf{x}_s) \right)^T \left(\Phi(\mathbf{x}_j) - \frac{1}{n} \sum_{s=1}^{n} \Phi(\mathbf{x}_s) \right)$$

$$= k(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{n}\sum_{s=1}^{n} k(\mathbf{x}_i, \mathbf{x}_s) - \frac{1}{n}\sum_{s=1}^{n} k(\mathbf{x}_s, \mathbf{x}_j) + \frac{1}{n^2}\sum_{s,t=1}^{n} k(\mathbf{x}_s, \mathbf{x}_t). \tag{33}$$

Algorithm 1 summarizes kernel PCA. The extra factor $1/\sqrt{\lambda_s}$ in the last step accounts for the normalization (32).

Algorithm 1 Kernel PCA

Require: $\{\mathbf{x}_i\}_{i=1}^n$ and the kernel $k(\cdot, \cdot)$

1: Compute the kernel matrix $K = (k(\mathbf{x}_i, \mathbf{x}_j)) \in \mathbb{R}^{n\times n}$

2: Compute the centered kernel matrix:

$$\tilde{K} = K - \frac{1}{n}EK - \frac{1}{n}KE + \frac{1}{n^2}EKE,$$

where E is a $n \times n$ matrix with all entries being 1.

3: Perform the standard eigen-decomposition for $\tilde{K}$:

$$\tilde{K}\tilde{\boldsymbol{\alpha}}_s = \lambda_s \tilde{\boldsymbol{\alpha}}_s,$$

with $\|\tilde{\boldsymbol{\alpha}}_s\|^2 = 1$.

4: **return** The s-th principal component is given by

$$p_s(\mathbf{x}) = \frac{1}{\sqrt{\lambda_s}}\sum_{i=1}^{n} \tilde{\alpha}_{s,i} k(\mathbf{x}_i, \mathbf{x}).$$

2.3 Autoencoder

Antoencoder is a general framework for dimension reduction. It consists of two components: the encoder f and the decoder g. The basic idea is summarized as follows:

$$f : \mathbb{R}^d \mapsto \mathbb{R}^k$$
$$g : \mathbb{R}^k \mapsto \mathbb{R}^d$$
$$f, g = \operatorname{argmin} \sum_{i=1}^{n} d(\mathbf{x}_i, g(f(\mathbf{x}_i))),$$

where $d(\cdot, \cdot)$ denotes a metric to measure the difference between $\mathbf{x}$ and its reconstruction $\tilde{\mathbf{x}} = g(f(\mathbf{x}))$. The following provides a semantic illustration

$$\mathbf{x} \in \mathbb{R}^d \xrightarrow{f} \mathbf{z} \in \mathbb{R}^k \xrightarrow{g} \tilde{\mathbf{x}} \in \mathbb{R}^d. \tag{34}$$

$\{\mathbf{z}_i = f(\mathbf{x}_i)\}$ are called the *latent codes*, the corresponding $\mathbb{R}^k$ with $k \ll d$ is called the *latent space*.

If we choose $f(\mathbf{x}) = V^T\mathbf{x}$, $g(\mathbf{z}) = U\mathbf{z}$, and $d(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|_2$. Then, the objective function becomes

$$\min_{U,V\in\mathbb{R}^{d\times k}} \sum_{i=1}^{n} \|\mathbf{x}_i - UV^T\mathbf{x}_i\|^2.$$

This recovers the standard PCA. However, autoencoder is more powerful than PCA since we can parameterize f, g using general nonlinear functions, such as deep neural networks.

2.4 Diffusion map

2.5 Random projection

The matrix $W = (w_{i,j}) \in \mathbb{R}^{k\times d}$ is said to be a random Gaussian matrix if $w_{i,j} \overset{iid}{\sim} \mathcal{N}(0,1)$. Consider the *random projection* $f_W : \mathbb{R}^d \mapsto \mathbb{R}^k$ given by

$$f_W(\mathbf{x}) := \frac{1}{\sqrt{k}} W\mathbf{x} \tag{35}$$

The following lemma states that this map is almost norm-preserving for reasonably large values of k.

Lemma 2.3 (Johnson-Lindenstrauss)**.** *Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be n samples in $\mathbb{R}^d$. Given any $\delta \in (0,1)$, let $k_\varepsilon = \frac{6\log(2n/\delta)}{\varepsilon^2}$. For any $k \geq k_\varepsilon$, with probability $1-\delta$ over the random sampling of W, we have*

$$\max_{i\in[n]} \left| \frac{\|\frac{1}{\sqrt{k}} W\mathbf{x}_i\|_2^2}{\|\mathbf{x}_i\|_2^2} - 1 \right| \leq \varepsilon \tag{36}$$

Proof. Without loss of generality, assume that $\|\mathbf{x}_i\|_2 = 1$. Let $W = (\mathbf{w}_1, \ldots, \mathbf{w}_k)^T$. Then $\mathbf{w}_j^T\mathbf{x} \sim \mathcal{N}(0,1)$ and

$$\mathbb{E}[\|\frac{1}{\sqrt{k}} W\mathbf{x}\|_2^2] = \mathbb{E}\,\frac{1}{k}\sum_{j=1}^k |\mathbf{w}_j^T\mathbf{x}|^2 = \frac{1}{k}\sum_{j=1}^k \mathbb{E}\,|\mathbf{w}_j^T\mathbf{x}|^2 = 1.$$

By the concentration inequality,

$$\mathbb{P}\Big\{ |\frac{1}{k}\sum_{j=1}^k |\mathbf{w}_j^T\mathbf{x}|^2 - 1| \geq \varepsilon \Big\} \leq 2e^{-\frac{k\varepsilon^2}{6}}.$$

Taking the union bound over the n points , we have

$$\mathbb{P}\Big\{ \max_{i\in[n]} |\frac{1}{k}\sum_{j=1}^k |\mathbf{w}_j^T\mathbf{x}|^2 - 1| \geq \varepsilon \Big\} \leq n2e^{-\frac{k\varepsilon^2}{6}}$$

Let the failure probability $2ne^{-k\varepsilon^2/6} \leq \delta$. We then have $k \geq \frac{6\log(2n/\delta)}{\varepsilon^2}$. □

The following lemma states that we can project n d-dimensional points into a $O(\frac{\log n}{\varepsilon^2})$-dimensional space and preserve all the pairwise distances within a factor of ε.

Lemma 2.4. *Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be n samples in $\mathbb{R}^d$. For any $\delta \in (0,1)$, if $k \geq \frac{6\log(2n^2/\delta)}{\varepsilon^2}$, we have with prob. $1-\delta$*

$$(1-\varepsilon)\|\mathbf{x}_i - \mathbf{x}_j\| \leq \|f_W(\mathbf{x}_i) - f_W(\mathbf{x}_j)\| \leq (1+\varepsilon)\|\mathbf{x}_i - \mathbf{x}_j\|.$$

Proof. Note that $f_W(\mathbf{x}_i) - f_W(\mathbf{x}_j) = f_W(\mathbf{x}_i - \mathbf{x}_j)$ since f_W is linear. Applying the Johnson-Lindenstrauss lemma to the set $\{\mathbf{x}_i - \mathbf{x}_j\}_{i,j}^n$ (n^2 points) completes the proof. □

From the proof, we can see that the distance-preserving property holds as long as each entry of the random matrix W is sampled from a sub-Gaussian distribution. This motivates the construction of sparse random projection, which can accelerate the evaluation of the random projection. A typical example is the one constructed by Achlioptas in [Achlioptas, 2003]:

$$w_{i,j} = \sqrt{3} \times \begin{cases} 1 & \text{with prob. } \frac{1}{6} \\ 0 & \text{with prob. } \frac{2}{3} \\ -1 & \text{with prob. } \frac{1}{6}. \end{cases} \tag{37}$$

Random projections are simple, easy to implement and do not require any special property for the data distribution. However, the reduced dimensionality using random projections is usually much higher than the ones obtained using the special properties of the data distribution, such as PCA.

3 Clustering

The objective of clustering is to decompose the dataset S into disjoint subsets:

$$S = \cup_{k=1}^{K} C_k, \quad C_k \cap C_j = \emptyset \quad \forall k \neq j,$$

such that

- points within each cluster are similar to each other.
- points in different clusters are dissimilar to each other.

Each subset is called a *cluster*. This problem is particularly relevant when the underlying distribution μ^* is multi-modal. Figure 9 shows two examples, where the data consist of two clusters.

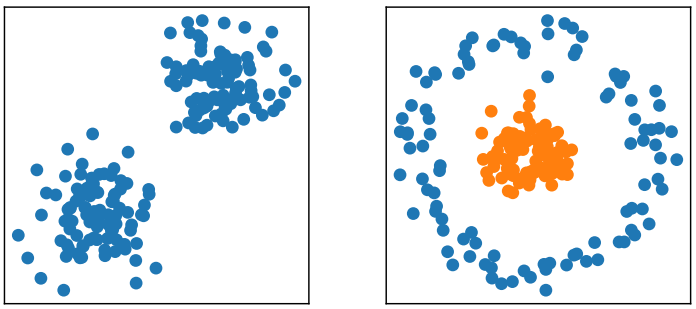

Figure 9: The data can be decomposed into two clusters.

Clustering has many applications. Here are two examples.

- Separating potential customers into different clusters in order to design the different products suited for each group of customers
- Image segmentation: Breaking up the image into different regions. Figure 10 shows how the image segmentation is used in autonomous driving.

To develop a clustering algorithm, the first issue is how to measure similarity or difference between two data points. If the data points live in $\mathbb{R}^d$, a natural measure is the Euclidean distance. For more complicated data sets such as texts or behavior data, it is less obvious how to define similarity measures.

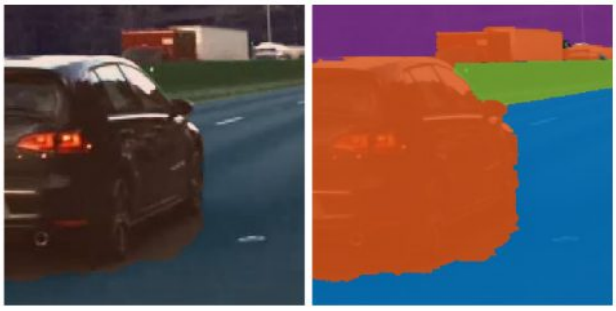

Figure 10: Autonomous driving. Clustering is used to help the system to identify and locate the vehicles and other objects on the road.

3.1 The k-means algorithm

The k-means algorithm is based on minimizing the intra-cluster distances. Assume that we intend to break the data into K clusters and let $C_1, C_2, \cdots, C_K$ be the clusters. In the k-means algorithm, the value of K needs to be specified beforehand. The objective function is defined to be:

$$I_1\left(C_1, C_2, \cdots, C_K\right) = \frac{1}{2}\sum_{k=1}^{K}\frac{1}{|C_k|}\sum_{\mathbf{x}_i,\mathbf{x}_j\in C_k}\|\mathbf{x}_i-\mathbf{x}_j\|^2, \tag{38}$$

where $|C_k|$ is the cardinality of C_k. Let

$$\boldsymbol{\alpha}_k = \frac{1}{|C_k|}\sum_{\mathbf{x}_j\in C_k}\mathbf{x}_j \tag{39}$$

be the center of the cluster C_k. A second natural objective function is given by

$$I_2\left(C_1, C_2, \cdots, C_K\right) = \sum_{k=1}^{K}\sum_{\mathbf{x}_i\in C_k}\|\mathbf{x}_i-\boldsymbol{\alpha}_k\|^2. \tag{40}$$

Lemma 3.1. $I_1 = I_2$.

Proof. Notice that

$$\begin{aligned}
\sum_{\mathbf{x}_i,\mathbf{x}_j\in C_k}\|\mathbf{x}_i-\mathbf{x}_j\|^2 &= \sum_{\mathbf{x}_i,\mathbf{x}_j\in C_k}\left(\|\mathbf{x}_i\|^2+\|\mathbf{x}_j\|^2-2\langle\mathbf{x}_i,\mathbf{x}_j\rangle\right)\\
&= 2|C_k|\sum_{\mathbf{x}_i\in C_k}\|\mathbf{x}_i\|^2-2\langle\sum_{\mathbf{x}_i\in C_k}\mathbf{x}_i,\sum_{\mathbf{x}_j\in C_k}\mathbf{x}_j\rangle\\
&= 2\big(|C_k|\sum_{\mathbf{x}_i\in C_k}\|\mathbf{x}_i\|^2-|C_k|^2\,\|\boldsymbol{\alpha}_k\|^2\big).\\
\sum_{\mathbf{x}_i\in C_k}\|\mathbf{x}_i-\boldsymbol{\alpha}_k\|^2 &= \sum_{\mathbf{x}_i\in C_k}\left(\|\mathbf{x}_i\|^2+\|\boldsymbol{\alpha}_k\|^2-2\langle\mathbf{x}_i,\alpha_k\rangle\right)\\
&= \sum_{\mathbf{x}_i\in C_k}\|\mathbf{x}_i\|^2-|C_k|\,\|\boldsymbol{\alpha}_k\|^2.
\end{aligned}$$

Hence,

$$I_1 = \frac{1}{2}\sum_{k=1}^{K}\frac{1}{|C_k|}\sum_{\mathbf{x}_i,\mathbf{x}_j\in C_k}\|\mathbf{x}_i-\mathbf{x}_j\|^2 = \frac{1}{2}\sum_{k}\frac{2C_k}{C_k}\sum_{\mathbf{x}_i\in C_k}\|\mathbf{x}_i-\mathbf{a}_k\|^2 = I_2.$$

□

The k-means algorithm, given in Algorithm 2, proceeds by alternating between two steps. In the assignment step, each data point is assigned to the cluster whose center is closest to that point. In the update step, the center position of each cluster is updated using the new assignments.

Algorithm 2 k-means algorithm

Require: Number of clusters K and the initialization: $\{\boldsymbol{\alpha}_k^{(0)}\}$

for $t = 1, 2, \ldots$ **do**

(1) Assignment step: For $k = 1, \ldots, K$, set

$$C_k^{(t+1)} := \{\mathbf{x}_i : k = \underset{j=1,\ldots,K}{\operatorname{argmin}}\|\mathbf{x}_i - \boldsymbol{\alpha}_j^{(t)}\|\}. \tag{41}$$

(2) Update step: $\boldsymbol{\alpha}_k^{(t+1)} = \frac{1}{|C_k^{(t+1)}|}\sum_{\mathbf{x}_j\in C_k^{(t+1)}}\mathbf{x}_j$.

end for

The k-means algorithm assumes implicitly that underlying data has some linear structure. For the data shown in the left figure of 9, k-means works very well. In fact, in this case k-means converges in a few iterations, see Figure 11 for a visualization of the convergence process. The right figure of 9 shows an example for which the k-means algorithm does not work. In this case we need some nonlinear clustering methods.

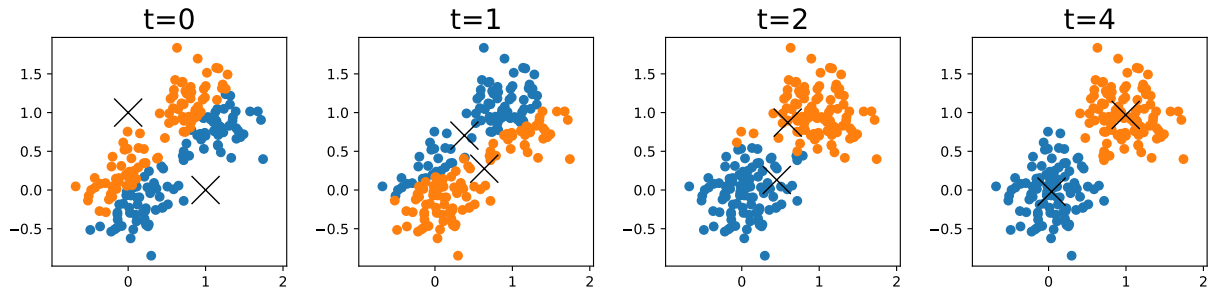

Figure 11: An example shows how k-means converges in a few steps.

this description is a bit too abstract

The following is a nice property of the k-means algorithm.

Lemma 3.2. *Let $\{C_k^{(t)}\}$ be the solution of the k-means algorithm at step t. Then we have*

$$I_2(C_1^{(t)}, \ldots, C_K^t) \le I_2(C_1^{(t-1)}, \ldots, C_K^{(t-1)}).$$

Proof. Let $\boldsymbol{\alpha}_k^{(t)}$ be the center of mass of the set $C_k^{(t)}$. For any $k \in [K]$, $\sum_{\mathbf{x}_i \in C_k^{(t)}} \|\mathbf{x}_i - \boldsymbol{\alpha}_k^{(t)}\|^2 \leq \sum_{\mathbf{x}_i \in C_k^{(t)}} \|\mathbf{x}_i - \boldsymbol{\alpha}\|^2$ for any $\boldsymbol{\alpha} \in \mathbb{R}^d$. Hence,

$$I_2(C_1^{(t)}, \ldots, C_K^{(t)}) = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k^{(t)}} \|\mathbf{x}_i - \boldsymbol{\alpha}_k^{(t)}\|^2 \leq \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k^{(t)}} \|\mathbf{x}_i - \boldsymbol{\alpha}_k^{(t-1)}\|^2.$$

The update procedure (41) implies that the new partition $\{C_k^{(t)}\}$ minimizes $\sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \boldsymbol{\alpha}_k^{(t-1)}\|$. Hence,

$$\begin{aligned} \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k^{(t)}} \|\mathbf{x}_i - \boldsymbol{\alpha}_k^{(t-1)}\|^2 &\leq \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k^{(t-1)}} \|\mathbf{x}_i - \boldsymbol{\alpha}_k^{(t-1)}\|^2 \\ &= I(C_1^{(t-1)}, \ldots, C_K^{(t-1)}). \end{aligned}$$

□

The preceding lemma shows that the objective function for the k-means algorithm decreases monotonically. It does not tell us whether the k-means algorithm converges to a global minimum. The objective function I_2 is non-convex and the convergence depends heavily on the initialization. Bad initializations may cause convergence to bad local minima (see the exercise for an example). In practice, one often needs to repeat the algorithm using different random initializations and pick the best solution.

3.2 Probabilistic Clustering

In k-means, we assign each sample to a single cluster. However, this type of description is not suited for the kind of data shown in Figure 12, where the clusters are not well-separated. A better description in this case is to define the "assignment" in a probabilistic fashion. Denote by p_k the probability that a given sample point belongs to the k-th cluster. Then $\sum_{k=1}^{K} p_k = 1$ and $p_k \geq 0$.

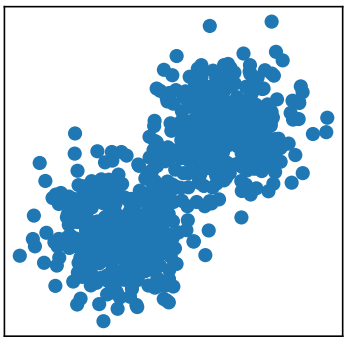

Figure 12: A motivating example where the probabilistic clustering method is preferred.

3.2.1 Gaussian mixture models

Before proceeding to the clustering method, let us first introduce a model for the kind of data distribution shown in Figure 12, the Gaussian mixture model (GMM). GMM assumes that the data are generated from the distribution

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k). \tag{42}$$

where $\sum_{k=1}^{K} \pi_k = 1$ and $\pi_k \geq 0$, $\mathcal{N}(\cdot|\boldsymbol{\mu}_k, \Sigma_k)$ is the Gaussian distribution. π_k and $\mathcal{N}(\cdot|\boldsymbol{\mu}_k, \Sigma_k)$ denote the weight and distribution for the k-th component in the mixture, respectively. Figure 13 shows an example of the density function of GMM with two mixtures.

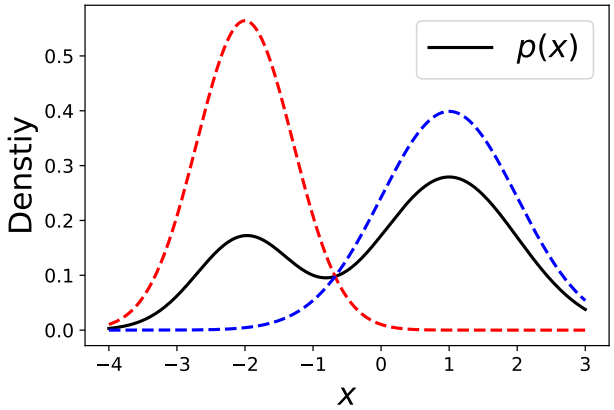

Figure 13: $p(x) = 0.3\mathcal{N}(-2, 0.5) + 0.7\mathcal{N}(1, 1)$

The direct problem of GMM is to draw samples from (42). This can be done as follows.

- Step 1: Randomly sample a $z \in \{1, \ldots, K\}$ with the probability distribution: $p(z = k) = \pi_k$.
- Step 2: Generate a sample from the z-th component: $\mathcal{N}(\cdot|\boldsymbol{\mu}_z, \Sigma_z)$.

The above procedure introduces a latent variable z such that

$$p(z = k) = \pi_k, \quad p(\mathbf{x}|z) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_z, \Sigma_z). \tag{43}$$

This leads to

$$p(\mathbf{x}) = \sum_{z} p(\mathbf{x}|z)p(z) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k).$$

A natural question is: Which cluster (mixture) does a given sample $\mathbf{x}$ belong to?

- In the k-means algorithm, $\mathbf{x}$ is assigned to the cluster with the index

$$k(\mathbf{x}) = \text{argmin}_{k \in [K]} \|\mathbf{x} - \boldsymbol{\alpha}_k\|.$$

- In GMM, the "label" is the posterior probability of z for a given $\mathbf{x}$:

$$\begin{aligned}\gamma_{\mathbf{x}}(k) := p(z=k|\mathbf{x}) &= \frac{p(\mathbf{x}|z=k)p(z=k)}{p(\mathbf{x})} \\ &= \frac{\pi_k \phi(\mathbf{x};\boldsymbol{\mu}_k,\Sigma_k)}{\sum_{z=1}^K \pi_z \phi(\mathbf{x};\boldsymbol{\mu}_z,\Sigma_z)},\end{aligned} \tag{44}$$

  where $\phi(\cdot;\boldsymbol{\mu}_k,\Sigma_k)$ denotes the density function of $\mathcal{N}(\boldsymbol{\mu}_k,\Sigma_k)$. The soft label $\gamma_{\mathbf{x}}(k)$ denotes the probability of $\mathbf{x}$ belonging to the k-th cluster.

Now we are ready to proceed to the inverse problem: Estimate the mixture model from a finite sample. To simplify the derivation, we focus on the simplest case when $d = 1$. The extension to the case when $d > 1$ is straightforward.

We begin with a brief review of the maximum likelihood estimator (MLE) for the Gaussian distribution. Let $x_i \overset{iid}{\sim} p(\cdot|\theta) := \mathcal{N}(\cdot;\mu,\sigma)$. Then log-likelihood is defined to be

$$\begin{aligned}L(\theta) = \log\prod_{i=1}^n p(x_i|\theta) &= \sum_{i=1}^n \log\left(\frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x_i-\mu)^2}{2\sigma}}\right) \\ &= -\sum_{i=1}^n \frac{|x_i-\mu|^2}{2\sigma} - \frac{n}{2}\log(2\pi\sigma).\end{aligned} \tag{45}$$

The MLE estimator can be found by solving:

$$\begin{aligned}\frac{\partial L}{\partial \mu} &= -\sum_{i=1}^n \frac{x_i-\mu}{\sigma} = 0 \quad \Rightarrow \quad \hat{\mu} = \frac{1}{n}\sum_{i=1}^n x_i \\ \frac{\partial L}{\partial \sigma} &= \frac{(x_i-\mu)^2}{2\sigma^2} - \frac{n}{2\sigma} = 0 \quad \Rightarrow \quad \hat{\sigma} = \frac{1}{n}\sum_{i=1}^n (x_i-\hat{\mu})^2.\end{aligned}$$

Next we turn to the MLE for GMM. Assume $x_i \overset{iid}{\sim} p(\cdot|\theta) := \sum_{k=1}^K \pi_k \mathcal{N}(\cdot;\mu_k,\sigma_k)$. The log-likelihood is defined by

$$L(\theta) = \log\prod_{i=1}^n p(x_i|\theta) = \sum_{i=1}^n \log\left(\sum_{k=1}^K \pi_k \frac{1}{\sqrt{2\pi\sigma_k}} e^{-\frac{(x_i-\mu_k)^2}{2\sigma_k}}\right). \tag{46}$$

The parameters in the MLE estimator are the solution of the following system of equations:

$$\begin{aligned}\frac{\partial L}{\partial \mu_k} &= -\sum_{i=1}^n \frac{\pi_k \phi(x_i;\mu_k,\sigma_k)}{\sum_z \pi_z \phi(x_i;\mu_z,\sigma_z)} \frac{x_i-\mu_k}{\sigma_k} = 0 \\ \frac{\partial L}{\partial \sigma_k} &= \sum_{i=1}^n \frac{\pi_k \phi(x_i;\mu_k,\sigma_k)}{\sum_z \pi_z \phi(x_i;\mu_z,\sigma_z)} \left(\frac{(x_i-\mu_k)^2}{2\sigma_k^2} - \frac{1}{2\sigma_k}\right) = 0\end{aligned}$$

However, it is difficult to obtain closed-form solutions due to the term

$$\frac{\pi_k \phi(x_i;\mu_k,\sigma_k)}{\sum_z \pi_z \phi(x_i;\mu_z,\sigma_z)}.$$

Notice that this term is exactly the soft label $\gamma_x(k)$ defined in Eq. (44). If we pretend that the $\gamma_{x_i}(k)$'s are known, the MLE estimator is given by

$$\begin{aligned}\hat{\mu}_k &= \frac{\sum_{i=1}^n \gamma_{x_i}(k) x_i}{\sum_{i=1}^n \gamma_{x_i}(k)} = \frac{1}{N_k} \sum_{i=1}^n \gamma_{x_i}(k) x_i \\ \hat{\sigma}_k &= \frac{1}{N_k} \sum_{i=1}^n \gamma_{x_i}(k)(x_i - \hat{\mu}_k)^2\end{aligned} \tag{47}$$

where $N_k = \sum_{i=1}^n \gamma_{x_i}(k)$. We can think of N_k as being the effective number of points assigned to the k-th cluster.

To estimate the mixture proportion $\{\pi_k\}$, consider the augmented Lagrangian $J = L(\theta) - \lambda \left(\sum_{k=1}^K \pi_k - 1 \right)$. Then,

$$\begin{aligned}\frac{\partial J}{\partial \pi_k} &= \frac{1}{\pi_k} \sum_{i=1}^n \frac{\pi_k \phi(x_i; \mu_k, \sigma_k)}{\sum_z \pi_z \phi(x_i; \mu_z, \sigma_z)} - \lambda = \frac{1}{\pi_k} N_k - \lambda = 0 \\ &\implies \pi_k = \frac{N_k}{\lambda}.\end{aligned}$$

Substituting this back to the constraint, we have

$$\sum_{k=1}^K \pi_k = \sum_{k=1}^K \frac{N_k}{\lambda} = \frac{n}{\lambda} = 1 \quad \Rightarrow \quad \lambda = n.$$

Hence, we have

$$\pi_k = \frac{N_k}{n} \tag{48}$$

However, we do not know the soft label $\gamma_{x_i}(k)$'s. One way to solve this problem is to estimate it using the current value of the parameters θ. This motivates the *Expectation-Maximization* (EM) algorithm for GMM, shown in Algorithm 3.

Algorithm 3 EM algorithm for Gaussian mixture models

Require: Initialization: $\theta^0 = \{\pi_k^0, \mu_k^0, \sigma_k^0\}$

for $t = 1, 2, \ldots$ **do**

(1) E-step: Estimate the soft labels using the current value of $\theta^t = \{\pi_k^t, \mu_k^t, \sigma_k^t\}$:

$$\gamma_{x_i}^t(k) = \frac{\pi_k^t \phi(x_i; \mu_k^t, \sigma_k^t)}{\sum_k \pi_k^t \phi(x_i; \mu_k^t, \sigma_k^t)}. \tag{49}$$

(2) M-step: Let $N_k^t = \sum_{i=1}^n \gamma_{x_i}^t(k)$. Update the parameters by

$$\begin{aligned} \mu_k^{t+1} &= \frac{1}{N_k^t} \sum_{i=1}^n \gamma_{x_i}^t(k) x_i \\ \sigma_k^{t+1} &= \frac{1}{N_k^t} \sum_{i=1}^n \gamma_{x_i}^t(k)(x_i - \mu_k^{t+1})^2 \\ \pi_k^{t+1} &= \frac{N_k^t}{n}. \end{aligned} \tag{50}$$

end for

return θ^t (the resulting parameters) and $\{\gamma_{x_i}^t\}$ (the soft labels).

3.3 General EM algorithm

The above idea can be extended to estimating parameters of a general latent variable model:

$$p(\mathbf{x}, \mathbf{z}|\theta) = p(\mathbf{x}|\mathbf{z}, \theta) p(\mathbf{z}|\theta), \tag{51}$$

where $\mathbf{x}$ is observed and $\mathbf{z}$ is the unobserved latent variable.

Let us first provide a heuristic derivation of the EM algorithm for estimating θ from only the observed samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$. We begin with the case of only one sample. The MLE is given by maximizing

$$L(\theta) = \log p(\mathbf{x}|\theta) = \log \left(\int p(\mathbf{x}, \mathbf{z}|\theta) \, \mathrm{d}\mathbf{z} \right). \tag{52}$$

This is computationally challenging due to the integral inside the logarithm. By the analogy with the derivation of EM for GMM, if $\mathbf{z}$ is also known, we only need to maximize $\log p(\mathbf{x}, \mathbf{z}|\theta)$. Thus the trouble lies in that $\mathbf{z}$ is not known yet. To overcome this difficulty, we sample the $\mathbf{z}$'s from $p(\mathbf{z}|\mathbf{x}, \theta^t)$ using the current value of θ^t. This step is analogous to the step of computing the soft labels using current value of the parameters in Algorithm 3. In expectation, we are maximizing

$$Q(\theta|\theta^t) := \mathbb{E}_{\mathbf{z}|\mathbf{x}, \theta^t}[\log p(\mathbf{x}, \mathbf{z}|\theta)]. \tag{53}$$

The general EM algorithm is given in Algorithm 4. Note that similar to the case of GMM, the maximization step usually has a closed-form solution for the most popular latent variable models. For this reason, the EM algorithm is usually quite easy to implement.

Algorithm 4 General EM algorithm

for $t = 1, 2, \ldots$ **do**
(1) E-step: $Q(\theta|\theta^t) = \mathbb{E}_{\mathbf{z}|\mathbf{x},\theta^t}[\log p(\mathbf{x}, \mathbf{z}|\theta)]$
(2) M-step: $\theta^{t+1} = \text{argmax}_\theta \, Q(\theta|\theta^t)$
end for

Analysis of the EM algorithm.

Since $p(\mathbf{x}|\theta)p(\mathbf{z}|\mathbf{x}, \theta) = p(\mathbf{x}, \mathbf{z}|\theta)$, we have the following decomposition of the log-likelihood:

$$\begin{aligned} L(\theta) &= \log p(\mathbf{x}|\theta) = \mathbb{E}_{\mathbf{z}|\mathbf{x},\theta^t}[\log p(\mathbf{x}|\theta)] \\ &= \mathbb{E}_{\mathbf{z}|\mathbf{x},\theta^t}[\log \frac{p(\mathbf{x}, \mathbf{z}|\theta)}{p(\mathbf{z}|\mathbf{x}, \theta)}] \\ &= \mathbb{E}_{\mathbf{z}|\mathbf{x},\theta^t}\left[\log \left(\frac{p(\mathbf{x}, \mathbf{z}|\theta)}{p(\mathbf{z}|\mathbf{x}, \theta^t)} \frac{p(\mathbf{z}|\mathbf{x}, \theta^t)}{p(\mathbf{z}|\mathbf{x}, \theta)}\right)\right] \\ &= \underbrace{\mathbb{E}_{\mathbf{z}|\mathbf{x},\theta^t}\left[\log \frac{p(\mathbf{x}, \mathbf{z}|\theta)}{p(\mathbf{z}|\mathbf{x}, \theta^t)}\right]}_{Q(\theta|\theta^t)} + \underbrace{\mathbb{E}_{\mathbf{z}|\mathbf{x},\theta^t}\left[\log \frac{p(\mathbf{z}|\mathbf{x}, \theta^t)}{p(\mathbf{z}|\mathbf{x}, \theta)}\right]}_{H(\theta|\theta^t)}, \end{aligned} \tag{54}$$

The first term is exactly the $Q(\theta|\theta^t)$ term that appears in the EM algorithm (up to a constant). The second term $H(\theta|\theta^t)$ is the KL divergence between $p(\mathbf{z}|\mathbf{x}, \theta)$ and $p(\mathbf{z}|\mathbf{x}, \theta^t)$, which is always non-negative. Hence the EM algorithm actually maximizes a lower bound of the log-likelihood $L(\theta)$ at each step. Moreover, when it approaches convergence, the lower bound becomes closer and closer to the true log-likelihood since $H(\theta^{t+1}|\theta^t) \to 0$ as $t \to \infty$.

Before proceeding to analyzing the convergence of the EM algorithm, we discuss some properties of the Q function. Since

$$Q(\theta|\theta') = \mathbb{E}_{\mathbf{z}|\mathbf{x},\theta'}\left[\log \frac{p(\mathbf{x}, \mathbf{z}|\theta)}{p(\mathbf{z}|\mathbf{x}, \theta')}\right],$$

we have

- $Q(\theta|\theta) = L(\theta)$ since $H(\theta|\theta) = 0$.
- $\nabla_\theta Q(\theta|\theta')|_{\theta'=\theta} = \nabla_\theta L(\theta)$ since

$$\begin{aligned} \nabla_\theta Q(\theta|\theta')|_{\theta'=\theta} = \mathbb{E}_{\mathbf{z}|\mathbf{x},\theta}[\nabla \log p(\mathbf{x}, \mathbf{z}|\theta)] &= \int \frac{\nabla p(\mathbf{x}, \mathbf{z}|\theta)}{p(\mathbf{x}, \mathbf{z}|\theta)} p(\mathbf{z}|\mathbf{x}, \theta) d\mathbf{z} \\ &= \int \frac{\nabla p(\mathbf{x}|\theta) p(\mathbf{z}|\mathbf{x}, \theta) + p(\mathbf{x}|\theta) \nabla p(\mathbf{z}|\mathbf{x}, \theta)}{p(\mathbf{x}|\theta)} d\mathbf{z} \\ &= \frac{\nabla p(\mathbf{x}|\theta)}{p(\mathbf{x}|\theta)} + \int \nabla p(\mathbf{z}|\mathbf{x}, \theta) d\mathbf{z} \\ &= \nabla \log p(\mathbf{x}|\theta) + \nabla \int p(\mathbf{z}|\mathbf{x}, \theta) d\mathbf{z} \\ &= \nabla_\theta L(\theta) \end{aligned}$$

The above implies that $Q(\cdot|\theta_t)$ is a lower bound of $L(\cdot)$ with the additional property that the function value and first order derivative are the same as those of $L(\cdot)$ at θ_t. Figure 14 provides an illustration of the relationship between $Q(\cdot|\theta_t)$ and $L(\cdot)$.

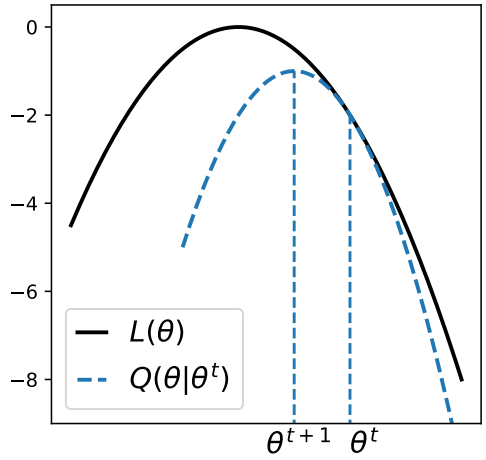

Figure 14: An illustration of EM algorithms.

Theorem 3.3 (Monotonicity)**.** *Let θ^t be the solution of Algorithm 4 at the t-th step. We have $L(\theta^{t+1}) \geq L(\theta^t)$.*

Proof. Using the decomposition (54), we have

$$\begin{aligned} L(\theta^{t+1}) - L(\theta^t) &= Q(\theta^{t+1}|\theta^t) + \underbrace{H(\theta^{t+1}|\theta^t)}_{\geq 0} - (Q(\theta^t|\theta^t) + H(\theta^t|\theta^t)) \\ &\geq Q(\theta^{t+1}|\theta^t) - Q(\theta^t|\theta^t) \geq 0, \end{aligned} \tag{55}$$

where the last inequality follows from the fact that $\theta^{t+1} = \operatorname{argmax}_\theta Q(\theta|\theta^t)$. □

Theorem 3.4 (Convergence to stationary points)**.** *Assume that $Q(\cdot|\theta')$ is C-smooth in the sense that for any θ'*

$$Q(\theta_2|\theta') \geq Q(\theta_1|\theta') + \langle \nabla Q(\theta_1|\theta'), \theta_2 - \theta_1 \rangle - \frac{C}{2}\|\theta_2 - \theta_1\|^2.$$

Then, we have

$$\min_{t=0,1,\ldots,T-1} \|\nabla L(\theta^t)\|^2 \leq \frac{1}{T}\sum_{t=0}^{T-1} \|\nabla L(\theta^t)\|^2 \leq \frac{2C(L(\theta_T) - L(\theta_0))}{T}.$$

Proof. Using the C-smoothness, we have

$$Q(\theta|\theta^t) \geq Q(\theta^t|\theta^t) + \langle \nabla Q(\theta^t|\theta^t), \theta - \theta^t \rangle - \frac{C}{2}\|\theta - \theta^t\|^2 \tag{56}$$

Using the fact that $\nabla Q(\theta^t|\theta^t) = \nabla L(\theta^t))$ and rearranging the right hand side of (56), we obtain

$$Q(\theta|\theta^t) \geq Q(\theta^t|\theta^t) + \frac{1}{2C}\|\nabla L(\theta^t)\|^2 - \frac{C}{2}\|\theta - \theta^t - \frac{1}{C}\nabla L(\theta^t)\|^2.$$

Let $\bar{\theta}^{t+1} = \theta^t + \frac{1}{C}\nabla L(\theta^t)$. Obviously we have

$$Q(\bar{\theta}^{t+1}|\theta^t) \geq Q(\theta^t|\theta^t) + \frac{1}{2C}\|\nabla L(\theta^t)\|^2.$$

Since $\theta^{t+1} = \operatorname{argmin}_\theta Q(\theta|\theta^t)$, we have

$$Q(\theta^{t+1}|\theta^t) \geq Q(\bar{\theta}^{t+1}|\theta^t) \geq Q(\theta^t|\theta^t) + \frac{1}{2C}\|\nabla L(\theta^t)\|^2.$$

Substituting this into Eq.(55), we have

$$\begin{aligned} L(\theta^{t+1}) - L(\theta^t) \geq Q(\theta^{t+1}|\theta^t) - Q(\theta^t|\theta^t) \geq \frac{1}{2C}\|\nabla L(\theta^t)\|^2. \\ \Rightarrow L(\theta^T) - L(\theta^0) = \sum_{t=1}^{T}(L(\theta^t) - L(\theta^{t-1})) \geq \frac{1}{2C}\sum_{t=0}^{T-1}\|\nabla L(\theta^t)\|^2. \end{aligned}$$

Thus, we complete the proof. □

Note that the above theorem only guarantees the convergence to stationary points.

References

[Achlioptas, 2003] Achlioptas, D. (2003). Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687.

[Boyd et al., 2004] Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

[Candès et al., 2011] Candès, E. J., Li, X., Ma, Y., and Wright, J. (2011). Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37.