Topic 7: Trending Data Models

Instructor: Daniel L. Pimentel-Alarcón

7.1 Subspaces

In many applications we observe data $\mathbf{x} \in \mathbb{R}^{\mathrm{D}}$ according to:

$$\mathbf{x} \;=\; \mathbf{U}\mathbf{c}^{\star} + \boldsymbol{\epsilon}, \tag{7.1}$$

where $\mathbf{U} \in \mathbb{R}^{\mathrm{D}\times\mathrm{R}}$ is a basis of a known subspace $\boldsymbol{\mathcal{U}}$, $\boldsymbol{\epsilon}$ represents noise, and $\mathbf{c}^{\star} \in \mathbb{R}^{\mathrm{R}}$ represents the *noiseless* coefficient of $\mathbf{x}$ with respect to (w.r.t.) $\mathbf{U}$.

Example 7.1 (Iris flowers dataset)**.** In the Iris Flowers dataset (also known as Fisher's or Anderson's) we can find the width and length (in centimeters) of the sepals and petals of 50 samples (flowers). Let $\mathbf{x}_{\mathrm{i}} \in \mathbb{R}^4$ be the vector containing the 4 values corresponding to the i^{th} flower. It turns out that the columns $\mathbf{x}_1, \ldots, \mathbf{x}_{50}$ lie mostly in a 2-dimensional subspace $\boldsymbol{\mathcal{U}}$ (see Figure 7.1 to build some intuition). This means that we can model this dataset as

$$\mathbf{x}_{\mathrm{i}} \;=\; \mathbf{U}\mathbf{c}_{\mathrm{i}}^{\star} + \boldsymbol{\epsilon}_{\mathrm{i}}$$

where $\mathbf{U} \in \mathbb{R}^{4\times 2}$ is a basis of $\boldsymbol{\mathcal{U}}$, $\mathbf{c}_{\mathrm{i}}^{\star} \in \mathbb{R}^2$ is the coefficient of $\mathbf{x}_{\mathrm{i}}$ in this basis, and $\boldsymbol{\epsilon}_{\mathrm{i}} \in \mathbb{R}^4$ represents small noise.

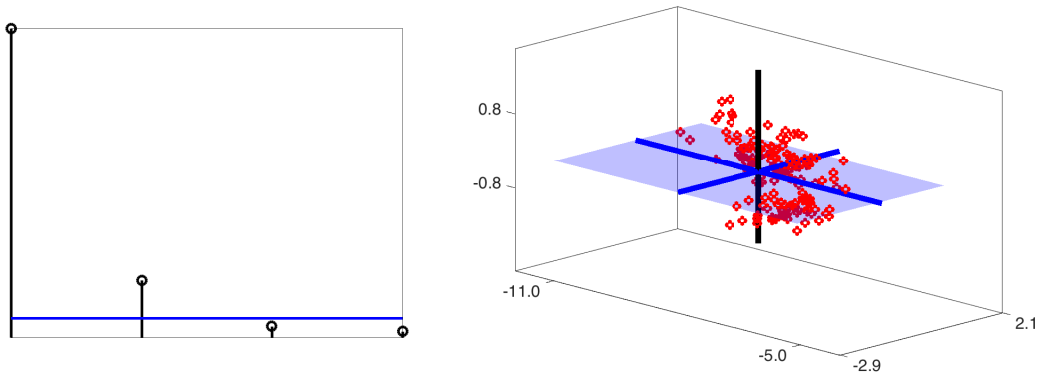

Figure 7.1: **Left:** Singular value decay of Iris Flowers dataset; blue line indicates 5% of the variance in the dataset. **Right:** Iris dataset lies mostly in 2-dimensional subspace. See Example 7.1.

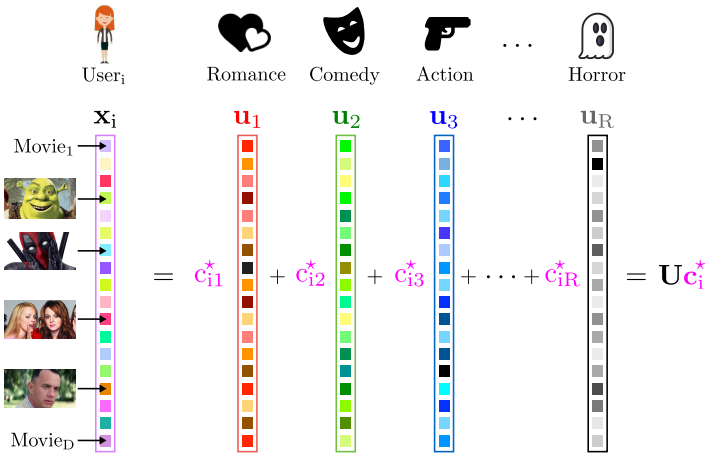

Figure 7.2: See Example 7.2

Example 7.2 (Netflix and recommender systems)**.** Netflix wants to know which movies you *would* like, so it can make you good recommendations, and you enjoy more your experience.

Consider the collection of all D movies in Netflix. Each movie has a component of romance, comedy, action, horror, etc. Let $\mathrm{R} \ll \mathrm{D}$ be the number of components, and let $\mathbf{u}_\mathrm{r} \in \mathbb{R}^\mathrm{D}$ be the vector containing the r^{th} *component* of all movies. For example, the j^{th} entry in $\mathbf{u}_1$ contains the amount of romance in the j^{th} movie.

On the other hand, we can think of each Netflix user as a combination of a romance-lover with a comedy-lover with an action-lover, etc. Let $\mathbf{c}^\star_\mathrm{i} \in \mathbb{R}^\mathrm{R}$ be the vector containing the *coefficients* of the i^{th} user. For example, the first entry in $\mathbf{c}^\star_\mathrm{i}$ indicates how much user i likes romance. See Figure 7.2 to build some intuition.

Hence we can model the vector of *preferences* $\mathbf{x}_\mathrm{i} \in \mathbb{R}^\mathrm{D}$ indicating how much user i likes each movie as

$$\mathbf{x}_\mathrm{i} \;=\; \mathbf{U}\mathbf{c}^\star_\mathrm{i} + \boldsymbol{\epsilon}_\mathrm{i},$$

where $\mathbf{U} \in \mathbb{R}^{\mathrm{D}\times\mathrm{R}}$ is the matrix formed with $\mathbf{u}_1, \ldots, \mathbf{u}_\mathrm{R}$ as columns, and $\boldsymbol{\epsilon}$ represents a small error term. This way, the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_\mathrm{N}$ of preferences of all users lie in the R-dimensional subspace $\boldsymbol{\mathcal{U}} = \mathrm{span}[\mathbf{U}]$. If Netflix can find the coefficients $\mathbf{c}^\star_\mathrm{i}$, it can make good predictions about what user i will like.

There is nothing special about movies. You can change movies for clothes, toys, food, etc., and the *features* {romance, comedy, etc.} for {color, size, etc.}, and you obtain a recommender system for any item. If you make good recommendations, users are more likely to buy. You can see why Amazon, Apple, Yelp, and all these big companies have great interest in this problem and are offering LOTS of money to people who work on this.

One of the first goals in these problems is to find $\mathbf{c}^\star$ based on the *noisy* observation $\mathbf{x}$ This can be posed as an estimation problem.

Example 7.3. Suppose we observe $\mathbf{x} \in \mathbb{R}^\mathrm{D}$ according to (7.1), where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I})$. Then

$$\boldsymbol{x} \;\sim\; \mathcal{N}(\mathbf{U}\mathbf{c}^\star, \sigma^2\mathbf{I}).$$

Our goal is to find $\mathbf{c}^\star$. Let's use our estimation techniques from before to find the maximum likelihood estimator (MLE):

$$\begin{aligned}\hat{\mathbf{c}}_{ML} &= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} \mathsf{p}(\mathbf{x}|\mathbf{c}) = \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} \underbrace{\frac{1}{(\sqrt{2\pi}\sigma)^{\mathrm{D}}}}_{\text{constant}} e^{-\frac{1}{2\sigma^2}(\mathbf{x}-\mathbf{U}\mathbf{c})^{\mathsf{T}}(\mathbf{x}-\mathbf{U}\mathbf{c})} \\ &= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} \log e^{-\frac{1}{2\sigma^2}(\mathbf{x}-\mathbf{U}\mathbf{c})^{\mathsf{T}}(\mathbf{x}-\mathbf{U}\mathbf{c})} = \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} -\underbrace{\frac{1}{2\sigma^2}}_{\text{constant}} (\mathbf{x}-\mathbf{U}\mathbf{c})^{\mathsf{T}}(\mathbf{x}-\mathbf{U}\mathbf{c}) \\ &= \arg\min_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} (\mathbf{x}-\mathbf{U}\mathbf{c})^{\mathsf{T}}(\mathbf{x}-\mathbf{U}\mathbf{c}) = \arg\min_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} \mathbf{x}^{\mathsf{T}}\mathbf{x} - 2\mathbf{c}^{\mathsf{T}}\mathbf{U}^{\mathsf{T}}\mathbf{x} + \mathbf{c}^{\mathsf{T}}\mathbf{U}^{\mathsf{T}}\mathbf{U}\mathbf{c}. \end{aligned} \tag{7.2}$$

We now proceed using elemental techniques from optimization: take derivative, set to zero and solve for the desired variable. To learn more about how to take derivatives w.r.t. vectors and matrices see *Old and new matrix algebra useful for statistics* by Thomas P. Minka. Taking derivative w.r.t. $\mathbf{c}$ and setting to zero, we obtain the following:

$$-2\mathbf{U}^{\mathsf{T}}\mathbf{x} + 2\mathbf{U}^{\mathsf{T}}\mathbf{U}\mathbf{c} = \mathbf{0},$$

and solving for $\mathbf{c}$:

$$\hat{\mathbf{c}}_{ML} = (\mathbf{U}^{\mathsf{T}}\mathbf{U})^{-1}\mathbf{U}^{\mathsf{T}}\mathbf{x}.$$

In Example 7.3 we found the MLE of $\mathbf{c}$ when the noise $\boldsymbol{\epsilon}$ was gaussian. In general, finding the MLE requires several lucky breaks, like:

- Knowing the distribution of the noise $\boldsymbol{\epsilon}$.
- Being able to write down the likelihood explicitly. What if $\boldsymbol{\epsilon} = \sum_{\mathrm{k}=1}^{\mathrm{K}} \boldsymbol{\epsilon}_{\mathrm{k}}$, where $\boldsymbol{\epsilon}_{\mathrm{k}} \sim \mathrm{Gamma}(\alpha_{\mathrm{k}}, \beta_{\mathrm{k}})$ with different β_{k}'s? Then we know that $\boldsymbol{x}$ has the distribution of the sum of Gammas, but this distribution has no known closed form.
- Being able to manipulate the likelihood algebraically so we can solve for its maximum in closed form or at least its derivatives, to use numerical optimization. Plus having a convex likelihood, so it has a unique maximum. For example, if $\boldsymbol{\epsilon}$ is a mixture of gaussians, the likelihood will generally be non-convex.

However, not everything is lost. Luckily, the model in (7.1) provides a very natural and general way to estimate $\mathbf{c}^\star$ as the coefficients of the projection of $\mathbf{x}$ onto the subspace $\boldsymbol{\mathcal{U}} = \mathrm{span}[\mathbf{U}]$. See Figure 7.3 to build some intuition. Recall that the projection $\hat{\mathbf{x}}_{\mathbf{U}} \in \boldsymbol{\mathcal{U}}$ of $\mathbf{x}$ is defined as

$$\hat{\mathbf{x}}_{\mathbf{U}} := \mathbf{x} \in \boldsymbol{\mathcal{U}} \text{ such that } \|\mathbf{x}-\hat{\mathbf{x}}\| \le \|\mathbf{x}-\widetilde{\mathbf{x}}\| \text{ for every } \widetilde{\mathbf{x}} \in \boldsymbol{\mathcal{U}},$$

and the coefficients $\hat{\mathbf{c}}_{\mathbf{U}}$ of $\hat{\mathbf{x}}_{\mathbf{U}}$ w.r.t. $\mathbf{U}$ are simply

$$\hat{\mathbf{c}}_{\mathbf{U}} := \mathbf{c} \in \mathbb{R}^{\mathrm{R}} \text{ such that } \hat{\mathbf{x}}_{\mathbf{U}} = \mathbf{U}\mathbf{c}.$$

Luckily, we can find $\hat{\mathbf{c}}_{\mathbf{U}}$ directly as:

$$\hat{\mathbf{c}}_{\mathbf{U}} = \arg\min_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} \|\mathbf{x}-\mathbf{U}\mathbf{c}\|_2^2 = \arg\min_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} (\mathbf{x}-\mathbf{U}\mathbf{c})^{\mathsf{T}}(\mathbf{x}-\mathbf{U}\mathbf{c}).$$

Notice that this is the same as (7.2), and hence

$$\hat{\mathbf{c}}_{\mathbf{U}} = (\mathbf{U}^{\mathsf{T}}\mathbf{U})^{-1}\mathbf{U}^{\mathsf{T}}\mathbf{x}.$$

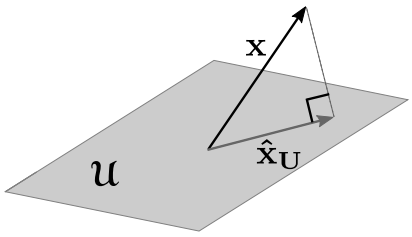

Figure 7.3: Projection $\hat{\mathbf{x}}_{\mathbf{U}}$ of vector $\mathbf{x}$ onto subspace $\mathcal{U}$.

Notice that $\hat{\mathbf{c}}_{\mathbf{U}}$ also provides an estimate of $\mathbf{x}^\star = \mathbf{U}\mathbf{c}^\star$, the *noiseless* counterpart of $\mathbf{x}$, via

$$\hat{\mathbf{x}}_{\mathbf{U}} \;=\; \mathbf{U}\hat{\mathbf{c}}_{\mathbf{U}} \;=\; \mathbf{U}(\mathbf{U}^\mathsf{T}\mathbf{U})^{-1}\mathbf{U}^\mathsf{T}\mathbf{x},$$

where we recognize $\mathbf{P}_{\mathbf{U}} = \mathbf{U}(\mathbf{U}^\mathsf{T}\mathbf{U})^{-1}\mathbf{U}^\mathsf{T} \in \mathbb{R}^{\mathrm{D}\times\mathrm{D}}$ to be the *projector operator* onto $\mathcal{U}$.

Example 7.4 (Linear regression)**.** In linear regression we want to infer how a variable y (e.g., glucose or cholesterol level) depends on an other collection of variables $\mathbf{x} \in \mathbb{R}^{\mathrm{M}}$ (e.g., diet, weight, income, education, sex, age) based on a sequence of *training* examples $\mathrm{y}_1, \ldots, \mathrm{y}_{\mathrm{N}}$. The main idea is to model y_i as a linear combination of $\mathbf{x}_\mathrm{i}$, i.e.,

$$\mathrm{y}_\mathrm{i} \;=\; \mathbf{x}_\mathrm{i}^\mathsf{T}\boldsymbol{\beta}^\star + \epsilon_\mathrm{i},$$

where ϵ_i models noise and the goal is to find $\boldsymbol{\beta}^\star \in \mathbb{R}^{\mathrm{M}}$, which determines the dependency of y on $\mathbf{x}$. Equivalently, in vector form we can write

$$\mathbf{y} \;=\; \mathbf{X}\boldsymbol{\beta}^\star + \boldsymbol{\epsilon},$$

where $\mathbf{y} = [\mathrm{y}_1 \;\cdots\; \mathrm{y}_{\mathrm{N}}]^\mathsf{T}$, $\mathbf{X}$ is the $\mathrm{N}\times\mathrm{M}$ matrix formed with $\{\mathbf{x}_\mathrm{i}^\mathsf{T}\}_{\mathrm{i}=1}^{\mathrm{N}}$ as rows and $\boldsymbol{\epsilon} = [\epsilon_\mathrm{i} \;\cdots\; \epsilon_{\mathrm{N}}]^\mathsf{T}$. Notice that this is equivalent to (7.1), only with slightly different notation: $\mathbf{y} \equiv \mathbf{x}$, $\mathbf{X} \equiv \mathbf{U}$, $\boldsymbol{\beta}^\star \equiv \mathbf{c}^\star$. It follows from our previous discussion that

$$\hat{\boldsymbol{\beta}} \;=\; (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y},$$

which is the well-known result from linear regression.

Remark 7.1. We point out that unlike $\hat{\mathbf{c}}_{ML}$, the form of $\hat{\mathbf{c}}_{\mathbf{U}}$ does not depend on the distribution of $\boldsymbol{\epsilon}$. In the case where $\boldsymbol{\epsilon}$ is gaussian, these two estimators $\hat{\mathbf{c}}_{ML}$ and $\hat{\mathbf{c}}_{\mathbf{U}}$ happen to be the same. However, in general this is not the case.

7.2 Sparsity

In the previous section we learned how to find the $\mathbf{c}$ that minimizes the error $\|\mathbf{x} - \mathbf{U}\mathbf{c}\|_2^2$. In many modern applications we have prior information that $\mathbf{c}$ should be *sparse*.

Figure 7.4: See Example 7.5. **Left:** Many astronomy images, like the Magellanic Clouds depicted here, are sparse. **Right:** The Wavelet transform of an MRI image is very sparse.

Example 7.5. Let $\mathbf{C} \in \mathbb{R}^{\sqrt{\mathrm{D}} \times \sqrt{\mathrm{D}}}$ be a *sparse* image. For example an astronomy image, or the Wavelet transform of an MRI image (see Figure 7.4). In many applications, rather than observing $\mathbf{C}$, we observe a *transformation* of $\mathbf{C}$. For example, telescopes will produce the Fourier transform of $\mathbf{C}$. This is equivalent to observing $\mathbf{x}$ as in (7.1), where $\mathbf{c} \in \mathbb{R}^{\mathrm{D}}$ is the *vectorized* version of $\mathbf{C}$ (i.e., the vector formed by stacking the columns in $\mathbf{C}$), and $\mathbf{U}$ determines the transformation. In cases like these, we know *a priori* that the desired solution $\mathbf{c}$ should be sparse.

We saw before that bayesian approaches *bias* our solutions towards our *prior* information. In this case, our prior information is that the solution should be sparse, so we will use a prior distribution on $\mathbf{c}$ that reflects this. One such popular distribution is the Laplacian distribution:

$$\mathsf{p}(\mathbf{c}) \;=\; \prod_{\mathrm{r}=1}^{\mathrm{R}} \frac{\lambda}{2} e^{-\lambda|\mathrm{c_r}|} \;=\; \left(\frac{\lambda}{2}\right)^{\mathrm{R}} e^{-\lambda \mathbf{c}^{\mathsf{T}} \mathrm{sign}(\mathbf{c})}.$$

Because of the shape of this density, a vector $\mathbf{c}$ with i.i.d. Laplacian entries is likely to have a lot of small values and a few large ones. This way, a Laplacian prior *favors* sparse solutions.

Notice that the Laplacian density has much heavier tails than the gaussian (see Figure 7.5; the Laplacian's exponential decay is $e^{-|\mathrm{c_r}|}$, while the gaussian's is $e^{-\mathrm{c_r^2}}$). This allows the Laplacian to produce *some* deviated values, i.e., mostly small values, and a few large ones. In contrast, since the gaussian density has lighter tails, is very unlikely to produce deviated values. A vector $\mathbf{c}$ with i.i.d. $\mathcal{N}(0, \sigma^2)$ entries will either have most entries be zero (if σ is smal) or most entries be large (if σ is large). Hence a gaussian is not a good *prior* to model sparsity. However, gaussian *likelihoods* are good to model noise. Luckily, as we will see in the next example, we can mix and match: we can use a Laplacian *prior* together with a gaussian *likelihood* in a bayesian setting in order to produce a sparse solution.

Example 7.6 (Laplacian prior)**.** Suppose we observe $\mathbf{x}$ as in (7.1), with $\mathrm{R} = \mathrm{D}$, $\mathbf{U}$ orthonormal, $\mathbf{c}^\star$ sparse, and $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2)$. Then

$$\boldsymbol{x} \;\sim\; \mathsf{p}(\mathbf{x}|\mathbf{c}) \;=\; \mathcal{N}(\mathbf{U}\mathbf{c}^\star, \sigma^2 \mathbf{I}),$$

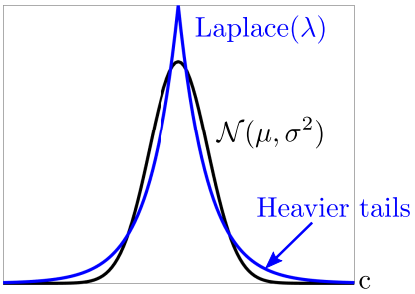

Figure 7.5: The Laplacian density is a good model for sparsity priors, as it would produce mostly small values, yet it would also produce a few large ones. This is because there is enough probability in its *heavy* tails. In contrast, the gaussian density is very unlikely to produce large values (deviated from the mean).

and we want to estimate $\mathbf{c}^\star$. Since we know $\mathbf{c}^\star$ is sparse, we can induce a Laplacian prior. Then our maximum a posteriori (MAP) estimator is

$$
\begin{aligned}
\hat{\mathbf{c}} &= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} \mathsf{p}(\mathbf{c}|\mathbf{x}) = \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} \log\big(\mathsf{p}(\mathbf{x}|\mathbf{c})\mathsf{p}(\mathbf{c})\big) \\
&= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} \log\Big(\frac{1}{\sqrt{(2\pi)^{\mathrm{D}}}} e^{-\frac{1}{2\sigma^2}(\mathbf{x}-\mathbf{U}\mathbf{c})^{\mathsf{T}}(\mathbf{x}-\mathbf{U}\mathbf{c})} \cdot \Big(\frac{\lambda}{2}\Big)^{\mathrm{R}} e^{-\lambda\mathbf{c}^{\mathsf{T}}\mathrm{sign}(\mathbf{c})}\Big) \\
&= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} \log e^{-\frac{1}{2\sigma^2}(\mathbf{x}-\mathbf{U}\mathbf{c})^{\mathsf{T}}(\mathbf{x}-\mathbf{U}\mathbf{c})-\lambda\mathbf{c}^{\mathsf{T}}\mathrm{sign}(\mathbf{c})} + \underbrace{\log\frac{\lambda^{\mathrm{R}}}{2^{\mathrm{R}}\sqrt{(2\pi)^{\mathrm{D}}}}}_{\text{constant } \kappa} \\
&= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} -\frac{1}{2\sigma^2}(\mathbf{x}-\mathbf{U}\mathbf{c})^{\mathsf{T}}(\mathbf{x}-\mathbf{U}\mathbf{c}) - \lambda\mathbf{c}^{\mathsf{T}}\mathrm{sign}(\mathbf{c}) + \kappa \\
&= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} -\frac{1}{2\sigma^2}\big(\mathbf{x}^{\mathsf{T}}\mathbf{x} - 2\mathbf{c}^{\mathsf{T}}\mathbf{U}^{\mathsf{T}}\mathbf{x} + \mathbf{c}^{\mathsf{T}}\underbrace{\mathbf{U}^{\mathsf{T}}\mathbf{U}}_{\mathbf{I}}\mathbf{c}\big) - \lambda\mathbf{c}^{\mathsf{T}}\mathrm{sign}(\mathbf{c}) + \kappa \\
&= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} -\frac{1}{2\sigma^2}\big(\mathbf{x}^{\mathsf{T}}\mathbf{x} - 2\mathbf{c}^{\mathsf{T}}\mathbf{U}^{\mathsf{T}}\mathbf{x} + \mathbf{c}^{\mathsf{T}}\mathbf{c}\big) - \lambda\mathbf{c}^{\mathsf{T}}\mathrm{sign}(\mathbf{c}) + \kappa \\
&= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} -\frac{1}{2\sigma^2}\big(\mathbf{x}^{\mathsf{T}}\underbrace{\mathbf{U}\mathbf{U}^{\mathsf{T}}}_{\mathbf{I}}\mathbf{x} - 2\mathbf{c}^{\mathsf{T}}\underbrace{\mathbf{U}^{\mathsf{T}}\mathbf{x}}_{=:\mathbf{b}} + \mathbf{c}^{\mathsf{T}}\mathbf{c}\big) - \lambda\mathbf{c}^{\mathsf{T}}\mathrm{sign}(\mathbf{c}) + \kappa \\
&= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} -\frac{1}{2\sigma^2}\big(\mathbf{b}^{\mathsf{T}}\mathbf{b} - 2\mathbf{c}^{\mathsf{T}}\mathbf{b} + \mathbf{c}^{\mathsf{T}}\mathbf{c}\big) - \lambda\mathbf{c}^{\mathsf{T}}\mathrm{sign}(\mathbf{c}) + \kappa \\
&= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} -\frac{1}{2\sigma^2}(\mathbf{b}-\mathbf{c})^{\mathsf{T}}(\mathbf{b}-\mathbf{c}) - \lambda\mathbf{c}^{\mathsf{T}}\mathrm{sign}(\mathbf{c}) + \kappa \\
&= \arg\max_{\mathbf{c}\in\mathbb{R}^{\mathrm{R}}} \sum_{\mathrm{r}=1}^{\mathrm{R}} \Big(-\frac{1}{2\sigma^2}(\mathrm{b_r}-\mathrm{c_r})^2 - \lambda|\mathrm{c_r}|\Big) + \kappa. \qquad (7.3)
\end{aligned}
$$

Next we need to study two cases:

(i) If $\mathrm{c_r} \neq 0$, the derivative of (7.3) w.r.t. $\mathrm{c_r}$ is $\frac{1}{\sigma^2}(\mathrm{b_r}-\mathrm{c_r}) - \lambda\mathrm{sign}(\mathrm{c_r})$. Setting to zero and solving for $\mathrm{c_r}$ we obtain $\mathrm{c_r} = \mathrm{b_r} - \lambda\sigma^2\mathrm{sign}(\mathrm{c_r})$. Notice that if $\mathrm{sign}(\mathrm{c_r}) \neq \mathrm{sign}(\mathrm{b_r})$, then $(\mathrm{b_r}-\mathrm{c_r})$ gets bigger, which is exactly the opposite of what we want. Thus we can replace $\mathrm{sign}(\mathrm{c_r})$ with $\mathrm{sign}(\mathrm{b_r})$ to

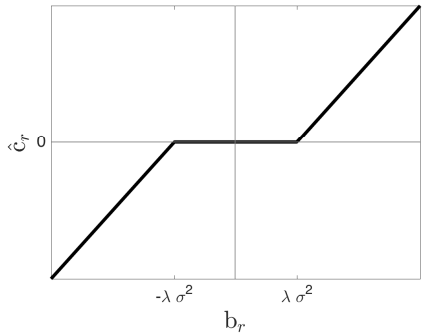

Figure 7.6: Soft-threshold function $\hat{c}_r = \text{sign}(b_r)\max(|b_r| - \lambda\sigma^2, 0)$. See Example 7.6.

obtain:

$$c_r \;=\; b_r - \lambda\sigma^2 \text{sign}(b_r).$$

Hence, if $c_r \neq 0$, the *cost* associated to the r^{th} term is

$$\left(\frac{1}{2\sigma^2}(b_r - c_r)^2 + \lambda|c_r|\right)\Big|_{c_r = b_r - \lambda\sigma^2\text{sign}(b_r)} \;=\; \frac{1}{2\sigma^2}\underbrace{\left(b_r - b_r + \lambda\sigma^2\text{sign}(b_r)\right)^2}_{\lambda^2\sigma^4} + \lambda\big|b_r - \lambda\sigma^2\text{sign}(b_r)\big|$$

$$= \;\frac{\lambda^2\sigma^2}{2} + \lambda\big|b_r - \lambda\sigma^2\text{sign}(b_r)\big|. \tag{7.4}$$

(ii) On the other hand, if $c_r = 0$, then the *cost* associated to the r^{th} term is

$$\left(\frac{1}{2\sigma^2}(b_r - c_r)^2 + \lambda|c_r|\right)\Big|_{c_r=0} \;=\; \frac{b_r^2}{2\sigma^2}. \tag{7.5}$$

Observing that

$$\begin{aligned} (7.4) \;&\geq\; (7.5) \qquad &&\text{if } |b_r| \leq \lambda\sigma^2, \\ (7.4) \;&<\; (7.5) \qquad &&\text{if } |b_r| > \lambda\sigma^2 \end{aligned}$$

it follows that the minimizer is

$$\hat{c}_r \;=\; \left\{ \begin{array}{ll} 0 & \text{if } |b_r| \leq \lambda\sigma^2 \\ b_r - \lambda\sigma^2\text{sign}(b_r) & \text{if } |b_r| < \lambda\sigma^2. \end{array} \right\} \;=\; \text{sign}(b_r)\max(|b_r| - \lambda\sigma^2, 0),$$

which is often known as the *soft-threshold* function (see Figure 7.6).

7.3 Mixtures and the EM algorithm

Previously we talked about *mixtures*, where

$$x_i \overset{iid}{\sim} \mathsf{p}(x|\boldsymbol{\theta}) \;=\; \sum_{k=1}^{K} \rho_k \;\mathsf{p}_k(x|\theta_k^\star),$$

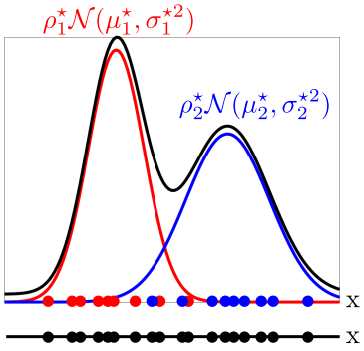

Figure 7.7: In a *mixture* each sample x_i is drawn according to one out of K distributions $p_k(x|\theta_k^\star)$, and we want to estimate $\boldsymbol{\theta}^\star = [\theta_1^\star \ \cdots \ \theta_K^\star]^T$. The challenge is that we don't know which distribution generated each x_i, and so we don't know which x_i's should be used to estimate $\theta_k^\star$. In this illustration we only observe the *black* points, which are drawn according to $p_k(x|\theta_k) = \mathcal{N}(\mu_k^\star, \sigma_k^{\star 2})$. If we knew which point is red and which one is blue, we could use only the red to estimate $\{\mu_1^\star, \sigma_1^\star\}$ and only the blue to estimate $\{\mu_2^\star, \sigma_2^\star\}$. The challenge is that we don't know the colors, and so we also have to estimate which distribution each point belongs to. Here $\rho_k^\star \in [0,1]$ models the fraction of points that correspond to the k^{th} distribution, which also has to be estimated.

that is, $x_i \overset{iid}{\sim} p_k(x|\theta_k^\star)$ with probability $\rho_k^\star$, $k = 1, \ldots, K$. See Figure 7.7 to build some intuition. The goal is to estimate $\boldsymbol{\theta}^\star = [\theta_1^\star \ \cdots \ \theta_K^\star]^T$. However, the MLE is not so easy to compute, and taking log does not simplify things because the log of a sum does not factor nicely. Furthermore, $p(x|\boldsymbol{\theta})$ is not even convex. This means that there may be multiple solutions (local maximums). Estimating mixtures is an active and challenging field of research.

There are several computational (iterative) methods to try to solve this problem, like the well-known gradient descent, or Newton's method. An other option is to use the *expectation-maximization* (EM) algorithm, which we will focus on. The intuition behind EM is that often things would greatly simplify if we had *side* information. For instance, consider the additional variable

$$z_i^\star = k \qquad \text{if } x_i \sim p_k(x|\theta_k^\star).$$

In words, $z_i^\star$ indicates which of the K distributions the i^{th} sample corresponds to. We don't know $\mathbf{z}^\star = [z_1^\star \ \cdots \ z_N^\star]^T$, but if we did, things would greatly simplify, because we could *split* our data into

$$\mathbf{X}_k^\star = \{x_i : z_i^\star = k\}, \qquad k = 1, \ldots, K,$$

and then compute the MLE of each $\theta_k^\star$ using only $\mathbf{X}_k^\star$.

Example 7.7. Consider a mixture of gaussians, i.e., $x_1, \ldots, x_N$ drawn i.i.d. according to

$$p(x|\boldsymbol{\theta}) = \sum_{k=1}^{K} \rho_k^\star \frac{1}{\sqrt{2\pi}\sigma_k^\star} e^{-\frac{1}{2}\left(\frac{x-\mu_k^\star}{\sigma_k^\star}\right)^2}.$$

This time our usual tricks from elementary optimization (taking derivatives and setting to zero) won't work, because the log term won't factor nicely. However, if we knew $\mathbf{z}$, we would be able to compute

the MLE of each $\theta_k^\star$ using only $\mathbf{X}_k^\star$ to obtain

$$\hat{\rho}_k \;=\; \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\{z_i^\star = k\}}, \qquad \hat{\mu}_k \;=\; \frac{1}{|\mathbf{X}_k^\star|}\sum_{x_i \in \mathbf{X}_k^\star} x_i, \qquad \hat{\sigma}_k^2 \;=\; \frac{1}{|\mathbf{X}_k^\star|}\sum_{x_i \in \mathbf{X}_k^\star} (x_i - \hat{\mu}_k)^2.$$

With this in mind, the EM algorithm incorporates the *missing data* $\mathbf{z}$ in the *complete-data* log-likelihood:

$$\log \mathsf{p}(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}).$$

Recall that we don't know $\mathbf{z}^\star$, so the idea is to average $\log \mathsf{p}(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})$ over $\mathbf{z}$ to obtain the *expected* complete-data log-likelihood

$$\mathsf{q}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t) \;:=\; \mathsf{E}_{\mathbf{z}|\mathbf{x},\hat{\boldsymbol{\theta}}_t}[\log \mathsf{p}(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})],$$

where the expectation is taken given $\mathbf{x}$ and the current estimate $\hat{\boldsymbol{\theta}}_t$. Intuitively, $\mathsf{q}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t)$ is the *simplified* likelihood that is easy to maximize. The EM algorithm, summarized in Algorithm 1, essentially consists on iteratively maximizing $\mathsf{q}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t)$.

Algorithm 1: Expectation-Maximization (EM)

Input: Data $x_1, \ldots, x_N$, number of gaussians K.
Initialize $\hat{\boldsymbol{\theta}}_t$, for example randomly.
Repeat until convergence

- **E-step.** Compute the expected complete log-likelihood:

$$\mathsf{q}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t) \;=\; \mathsf{E}_{\mathbf{z}|\mathbf{x},\hat{\boldsymbol{\theta}}_t}[\log \mathsf{p}(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})].$$

- **M-step.** Maximize the expected complete log-likelihood:

$$\hat{\boldsymbol{\theta}}_{t+1} := \arg\max_{\boldsymbol{\theta}} \; \mathsf{q}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t).$$

Output: $\hat{\boldsymbol{\theta}}_{t+1}$.

The EM algorithm is attractive when q is easily computed and maximized.

Example 7.8. Consider a mixture of gaussians, as in Example 7.7. We can write $\mathsf{p}(x_i, z_i|\boldsymbol{\theta})$ as

$$\mathsf{p}(x_i, z_i|\boldsymbol{\theta}) \;=\; \mathsf{p}(x_i|z_i, \boldsymbol{\theta})\mathsf{p}(z_i|\boldsymbol{\theta}) \;=\; \prod_{k=1}^{K} \left(\rho_k \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{1}{2}\left(\frac{x_i - \mu_k}{\sigma_k}\right)^2} \right)^{\mathbb{1}_{\{z_i = k\}}}.$$

It follows that

$$\begin{aligned}\log \mathsf{p}(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) &= \log \prod_{\mathrm{i}=1}^{\mathrm{N}} \prod_{\mathrm{k}=1}^{\mathrm{K}} \left(\rho_{\mathrm{k}} \frac{1}{\sqrt{2\pi}\sigma_{\mathrm{k}}} e^{-\frac{1}{2}\left(\frac{\mathrm{x}_i - \mu_{\mathrm{k}}}{\sigma_{\mathrm{k}}}\right)^2} \right)^{\mathbb{1}_{\{\mathrm{z}_\mathrm{i}=\mathrm{k}\}}} \\ &= \sum_{\mathrm{i}=1}^{\mathrm{N}} \sum_{\mathrm{k}=1}^{\mathrm{K}} \log \left(\rho_{\mathrm{k}} \frac{1}{\sqrt{2\pi}\sigma_{\mathrm{k}}} e^{-\frac{1}{2}\left(\frac{\mathrm{x}_i - \mu_{\mathrm{k}}}{\sigma_{\mathrm{k}}}\right)^2} \right)^{\mathbb{1}_{\{\mathrm{z}_\mathrm{i}=\mathrm{k}\}}} \\ &= \sum_{\mathrm{i}=1}^{\mathrm{N}} \sum_{\mathrm{k}=1}^{\mathrm{K}} \mathbb{1}_{\{\mathrm{z}_\mathrm{i}=\mathrm{k}\}} \log \rho_{\mathrm{k}} \frac{1}{\sqrt{2\pi}\sigma_{\mathrm{k}}} e^{-\frac{1}{2}\left(\frac{\mathrm{x}_i - \mu_{\mathrm{k}}}{\sigma_{\mathrm{k}}}\right)^2}.\end{aligned}$$

Taking expectation we have:

$$\mathsf{q}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t) = \mathsf{E}_{\mathrm{z}_\mathrm{i}|\mathrm{x}_\mathrm{i},\hat{\boldsymbol{\theta}}_t}[\mathsf{p}(\mathrm{x}_\mathrm{i}, \mathrm{z}_\mathrm{i}|\boldsymbol{\theta})] = \sum_{\mathrm{i}=1}^{\mathrm{N}} \sum_{\mathrm{k}=1}^{\mathrm{K}} \mathsf{E}_{\mathrm{z}_\mathrm{i}|\mathrm{x}_\mathrm{i},\hat{\boldsymbol{\theta}}_t}[\mathbb{1}_{\{\mathrm{z}_\mathrm{i}=\mathrm{k}\}}] \log \rho_{\mathrm{k}} \frac{1}{\sqrt{2\pi}\sigma_{\mathrm{k}}} e^{-\frac{1}{2}\left(\frac{\mathrm{x}_i - \mu_{\mathrm{k}}}{\sigma_{\mathrm{k}}}\right)^2},$$

where

$$\mathsf{E}_{\mathrm{z}_\mathrm{i}|\mathrm{x}_\mathrm{i},\hat{\boldsymbol{\theta}}_t}[\mathbb{1}_{\{\mathrm{z}_\mathrm{i}=\mathrm{k}\}}] = \mathsf{p}(\mathrm{z}_\mathrm{i} = \mathrm{k}|\mathrm{x}_\mathrm{i}, \hat{\boldsymbol{\theta}}_t) = \frac{\mathsf{p}(\mathrm{x}_\mathrm{i}|\mathrm{z}_\mathrm{i} = \mathrm{k}, \hat{\boldsymbol{\theta}}_t)\mathsf{p}(\mathrm{z}_\mathrm{i} = \mathrm{k}|\hat{\boldsymbol{\theta}}_t)}{\mathsf{p}(\mathrm{x}_\mathrm{i}|\hat{\boldsymbol{\theta}}_t)} = \frac{\hat{\rho}_{\mathrm{k}_t} \frac{1}{\sqrt{2\pi}\hat{\sigma}_{\mathrm{k}_t}} e^{-\frac{1}{2}\left(\frac{\mathrm{x}_i - \hat{\mu}_{\mathrm{k}_t}}{\hat{\sigma}_{\mathrm{k}_t}}\right)^2}}{\sum_{\mathrm{k}=1}^{\mathrm{K}} \hat{\rho}_{\mathrm{k}_t} \frac{1}{\sqrt{2\pi}\hat{\sigma}_{\mathrm{k}_t}} e^{-\frac{1}{2}\left(\frac{\mathrm{x}_i - \hat{\mu}_{\mathrm{k}_t}}{\hat{\sigma}_{\mathrm{k}_t}}\right)^2}}$$

can be easily computed. At this point we have already completed the E-step. To obtain $\hat{\boldsymbol{\theta}}_{t+1} = \arg\max_{\boldsymbol{\theta}} \mathsf{q}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t)$ in the M-step, we can use our usual tricks: take derivative, set to zero, and solve for the maximizer, which yields:

$$\begin{aligned}\hat{\rho}_{\mathrm{k}_{t+1}} &= \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathsf{p}(\mathrm{z}_\mathrm{i} = \mathrm{k}|\mathrm{x}_\mathrm{i}, \hat{\boldsymbol{\theta}}_t), \\ \hat{\mu}_{\mathrm{k}_{t+1}} &= \frac{1}{\mathrm{N}\hat{\rho}_{\mathrm{k}_{t+1}}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathsf{p}(\mathrm{z}_\mathrm{i} = \mathrm{k}|\mathrm{x}_\mathrm{i}, \hat{\boldsymbol{\theta}}_t)\ \mathrm{x}_\mathrm{i}, \\ \hat{\sigma}^2_{\mathrm{k}_{t+1}} &= \frac{1}{\mathrm{N}\hat{\rho}_{\mathrm{k}_{t+1}}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathsf{p}(\mathrm{z}_\mathrm{i} = \mathrm{k}|\mathrm{x}_\mathrm{i}, \hat{\boldsymbol{\theta}}_t)\ (\mathrm{x}_\mathrm{i} - \hat{\mu}_{\mathrm{k}_{t+1}})^2.\end{aligned}$$

Exercise 7.1. Derive the EM algorithm for a mixture of *multivariate* gaussians.

Remark 7.2. The EM algorithm is useful only if $\mathsf{q}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t)$ is easy to compute and maximize. Otherwise we would need to maximize an ugly function, which is what we are trying to avoid. Moreover, we would need to do so at each step! Also, determining which side information to use to simplify the problem involves a bit of art, and needs to be tailored case by case. Luckily, there are many cases where this can be done.

The EM algorithm is widely used to solve non-convex problems with missing data (in our case, the *missing*

data is $\mathbf{z}$). One of the reasons of EM's great success is that its estimates are guaranteed to get better with each iteration.

Proposition 7.1 (EM-convergence)**.** Let $\hat{\boldsymbol{\theta}}_t$ be the estimate produced by the EM algorithm at time t. Then

$$\mathsf{p}(\mathbf{x}|\hat{\boldsymbol{\theta}}_{t+1}) \;\geq\; \mathsf{p}(\mathbf{x}|\hat{\boldsymbol{\theta}}_t).$$

Proof. We want to show that $\mathsf{p}(\mathbf{x}|\hat{\boldsymbol{\theta}}_{t+1}) \;\geq\; \mathsf{p}(\mathbf{x}|\hat{\boldsymbol{\theta}}_t)$, or equivalently, that

$$\log \mathsf{p}(\mathbf{x}|\hat{\boldsymbol{\theta}}_{t+1}) - \log \mathsf{p}(\mathbf{x}|\hat{\boldsymbol{\theta}}_t) \;\geq\; 0.$$

We know from elementary probability that $\mathsf{p}(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) = \mathsf{p}(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})\mathsf{p}(\mathbf{x}|\boldsymbol{\theta})$, so

$$\begin{aligned}
\log \mathsf{p}(\mathbf{x}|\hat{\boldsymbol{\theta}}_{t+1}) - \log \mathsf{p}(\mathbf{x}|\hat{\boldsymbol{\theta}}_t) \;&=\; \log \mathsf{p}(\mathbf{x}, \mathbf{z}|\hat{\boldsymbol{\theta}}_{t+1}) - \log \mathsf{p}(\mathbf{z}|\mathbf{x}, \hat{\boldsymbol{\theta}}_{t+1}) - \log \mathsf{p}(\mathbf{x}, \mathbf{z}|\hat{\boldsymbol{\theta}}_t) + \log \mathsf{p}(\mathbf{z}|\mathbf{x}, \hat{\boldsymbol{\theta}}_t) \\
&=\; \log \mathsf{p}(\mathbf{x}, \mathbf{z}|\hat{\boldsymbol{\theta}}_{t+1}) - \log \mathsf{p}(\mathbf{x}, \mathbf{z}|\hat{\boldsymbol{\theta}}_t) + \log \frac{\mathsf{p}(\mathbf{z}|\mathbf{x}, \hat{\boldsymbol{\theta}}_t)}{\mathsf{p}(\mathbf{z}|\mathbf{x}, \hat{\boldsymbol{\theta}}_{t+1})}.
\end{aligned}$$

Next notice that $\log \mathsf{p}(\mathbf{x}|\boldsymbol{\theta})$ is not a function of $\mathbf{z}$, so taking expectations on both sides,

$$\begin{aligned}
\log \mathsf{p}(\mathbf{x}|\hat{\boldsymbol{\theta}}_{t+1}) - \log \mathsf{p}(\mathbf{x}|\hat{\boldsymbol{\theta}}_t) \;&=\; \underbrace{\mathsf{E}_{\mathbf{z}|\mathbf{x},\hat{\boldsymbol{\theta}}_t}\left[\log \mathsf{p}(\mathbf{x}, \mathbf{z}|\hat{\boldsymbol{\theta}}_{t+1})\right]}_{\mathsf{q}(\hat{\boldsymbol{\theta}}_{t+1}|\hat{\boldsymbol{\theta}}_t)} - \underbrace{\mathsf{E}_{\mathbf{z}|\mathbf{x},\hat{\boldsymbol{\theta}}_t}\left[\log \mathsf{p}(\mathbf{x}, \mathbf{z}|\hat{\boldsymbol{\theta}}_t)\right]}_{\mathsf{q}(\hat{\boldsymbol{\theta}}_t|\hat{\boldsymbol{\theta}}_t)} + \underbrace{\mathsf{E}_{\mathbf{z}|\mathbf{x},\hat{\boldsymbol{\theta}}_t}\left[\log \frac{\mathsf{p}(\mathbf{z}|\mathbf{x}, \hat{\boldsymbol{\theta}}_t)}{\mathsf{p}(\mathbf{z}|\mathbf{x}, \hat{\boldsymbol{\theta}}_{t+1})}\right]}_{\text{KL divergence} \geq 0} \\
&\geq\; \mathsf{q}(\hat{\boldsymbol{\theta}}_{t+1}|\hat{\boldsymbol{\theta}}_t) - \mathsf{q}(\hat{\boldsymbol{\theta}}_t|\hat{\boldsymbol{\theta}}_t) \;\geq\; 0,
\end{aligned}$$

where the last inequality follows because $\hat{\boldsymbol{\theta}}_{t+1} = \arg\max_{\boldsymbol{\theta}} \; \mathsf{q}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t)$ by definition, so

$$\mathsf{q}(\hat{\boldsymbol{\theta}}_{t+1}|\hat{\boldsymbol{\theta}}_t) \;=\; \max_{\boldsymbol{\theta}} \mathsf{q}(\boldsymbol{\theta}|\hat{\boldsymbol{\theta}}_t) \;\geq\; \mathsf{q}(\hat{\boldsymbol{\theta}}_t|\hat{\boldsymbol{\theta}}_t).$$

□

7.4 Subspace Clustering

In subspace clustering we observe column vectors $\mathbf{x}_1, \ldots, \mathbf{x}_\mathrm{N}$ lying in the union of K R-dimensional subspaces of $\mathbb{R}^\mathrm{D}$, $\mathcal{U}_1^\star, \ldots, \mathcal{U}_\mathrm{K}^\star$. The goal is to estimate the underlying subspaces from $\mathbf{x}_1, \ldots, \mathbf{x}_\mathrm{N}$, and to cluster the columns according to the subspaces. See Figure **??** to build some intuition.

This is a problem with active research that has attracted a lot of attention in recent years due to its wide range of applications, including computer vision, network inference and recommenders systems, among others []. We will now analyze it in light of the topics we discussed before.

7.4.1 Subspace Clustering as a Mixture

First, subspace clustering can be posed as a mixture problem

$$\mathbf{x}_\mathrm{i} \overset{iid}{\sim} \sum_{\mathrm{k}=1}^{\mathrm{K}} \rho_\mathrm{k} \, \mathcal{N}(\mathbf{0}, \mathbf{U}_\mathrm{k}\mathbf{U}_\mathrm{k}^\mathsf{T}),$$

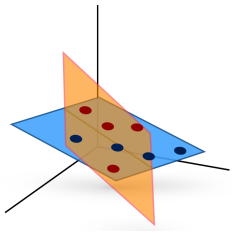

Figure 7.8: Some vectors (blue) lie in one subspace, other vectors (red) lie in an other subspace.

where $\mathbf{U}_k \in \mathbb{R}^{D \times R}$ is a basis of $\boldsymbol{\mathcal{U}}_k$.

Exercise 7.2. Derive an EM algorithm for subspace clustering.

7.4.2 Subspace Clustering using Sparsity

We will now demonstrate the power of sparsity for clustering applications. In particular we will discuss the main principle behind the state-of-the-art *sparse subspace clustering* algorithm (SSC) []. The key idea is to try to write each $\mathbf{x}_i$ as a linear combination of *other* columns from the dataset, i.e.,

$$\mathbf{x}_i \;=\; \mathbf{X}_i \mathbf{c}_i, \tag{7.6}$$

where $\mathbf{X}_i$ is the $D \times (N-1)$ matrix containing all the columns except $\mathbf{x}_i$, and $\mathbf{c}_i \in \mathbb{R}^{N-1}$ is the vector of *coefficients* of $\mathbf{x}_i$ with respect to $\mathbf{X}_i$. In general, there may be infinitely many vectors $\mathbf{c}_i$ satisfying (7.6). However, if $\mathbf{x}_i \in \boldsymbol{\mathcal{U}}_k$, and $\mathbf{X}_i$ contains a basis of $\boldsymbol{\mathcal{U}}_k$, then we can write $\mathbf{x}_i$ as a *sparse* linear combination of the columns in $\mathbf{X}_i$. For example, if $i = N$ and the first R columns of $\mathbf{X}_i$ span $\boldsymbol{\mathcal{U}}_k$, then we can write

$$\mathbf{x}_i \;=\; \mathbf{X}_i \mathbf{c}_i^\star \;=\; \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_R & \mathbf{x}_{R+1} & \cdots & \mathbf{x}_{N-1} \end{bmatrix} \begin{bmatrix} c_{i_1}^\star \\ \vdots \\ c_{i_R}^\star \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{7.7}$$

for some $[c_{i_1}^\star \;\cdots\; c_{i_R}^\star]^\top$, containing the coefficients of $\mathbf{x}_i$ with respect to the basis $[\mathbf{x}_1 \;\ldots\; \mathbf{x}_R]$.

Notice that if $\mathbf{X}_i$ contains multiple basis of $\boldsymbol{\mathcal{U}}_k$ (which is generally the case in subspace clustering applications), there may be multiple sparse *representations* of $\mathbf{x}_i$. Continuing with our example, if $[\mathbf{x}_2 \;\ldots\; \mathbf{x}_{R+1}]$ are *also* a basis of $\boldsymbol{\mathcal{U}}_k$, then we could *also* write

$$\mathbf{x}_i \;=\; \mathbf{X}_i \mathbf{c}_i^\star \;=\; \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{R+1} & \mathbf{x}_{R+2} & \cdots & \mathbf{x}_{N-1} \end{bmatrix} \begin{bmatrix} 0 \\ c_{i_2}^\star \\ \vdots \\ c_{i_{R+1}}^\star \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

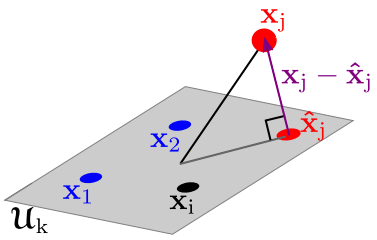

Figure 7.9: Suppose we find that $\mathbf{x}_i$ can be written as a linear combination of $\mathbf{x}_1, \mathbf{x}_2$, as in (7.7). This means that $\mathbf{x}_i, \mathbf{x}_1, \mathbf{x}_2$ lie in the same subspace $\mathcal{U}_k$. But more importantly, this means that $\mathbf{x}_1, \mathbf{x}_2$ are a basis of $\mathcal{U}_k$. In other words, we have already identified $\mathcal{U}_k$. We can then find *all* the columns that lie in $\mathcal{U}_k$ by simply projecting onto $\mathcal{U}_k$ and checking residuals. If a column lies in $\mathcal{U}_k$, like $\mathbf{x}_i$, then its residual $\mathbf{x}_i - \hat{\mathbf{x}}_j$ will be zero. If a column does not lie in $\mathcal{U}_k$, like $\mathbf{x}_j$, its residual $\mathbf{x}_j - \hat{\mathbf{x}}_j$ will be nonzero.

for some *other* $[c^\star_{i_2} \ \cdots \ c^\star_{i_{R+1}}]^T$, containing the coefficients of $\mathbf{x}_i$ with respect to the basis $[\mathbf{x}_2 \ \ldots \ \mathbf{x}_{R+1}]$.

Luckily, regardless of the particular *sparse* $\mathbf{c}_i^\star$, the nonzero coefficients of $\mathbf{c}_i^\star$ determine R columns in $\mathbf{X}_i$ that correspond to the same subspace as $\mathbf{x}_i$. But more importantly, these R columns form a basis of $\mathcal{U}_k$. In other words, we have already identified $\mathcal{U}_k$, and hence we can identify *all* the columns in $\mathbf{X}$ that lie in $\mathcal{U}_k$ (by simply projecting onto $\mathcal{U}_k$, and checking residuals; see Figure 7.9 to build some intuition). Hence the goal is to find *a sparse* $\mathbf{c}_i^\star$. After that, we can *prune* $\mathbf{X}$ (i.e., remove all the columns corresponding to $\mathcal{U}_k$) and repeat this procedure to find the columns corresponding to each of the subspaces (which is what we ultimately want).

So the goal is to find *a sparse* $\mathbf{c}_i^\star$ such that $\mathbf{x}_i = \mathbf{X}_i\mathbf{c}_i^\star$. Ideally, this could be done by solving:

$$\hat{\mathbf{c}}_i \ = \ \underset{\mathbf{c}_i : \|\mathbf{c}_i\|_0 = R}{\arg\min} \ \|\mathbf{x}_i \ - \ \mathbf{X}_i\mathbf{c}_i\|,$$

or equivalently,

$$\hat{\mathbf{c}}_i \ = \ \underset{\mathbf{c}_i}{\arg\min} \ \|\mathbf{x}_i \ - \ \mathbf{X}_i\mathbf{c}_i\| \ + \ \|\mathbf{c}_i\|_0, \tag{7.8}$$

where $\|\cdot\|_0$ is the so-called ℓ_0-norm (which is actually *not* a norm! Can you show this?), defined as the number of nonzero entries in a vector. Sadly, the ℓ_0-norm is non-convex, and minimizing it is NP-hard.

Whenever one wants to minimize an ugly, non-convex function f (as in (7.8)), it is often practical to minimize *the next best convex thing* instead, i.e., minimize an other function g that is both convex and similar to f. This is usually called *convex relaxation*. In our case, $g = \ell_1$-norm (defined as the sum of the absolute values of the entries in a vector) is both convex and similar to $f = \ell_0$-norm. Hence, instead of (7.8), we can solve

$$\hat{\mathbf{c}}_i \ = \ \underset{\mathbf{c}_i}{\arg\min} \ \|\mathbf{x}_i \ - \ \mathbf{X}_i\mathbf{c}_i\| \ + \ \lambda\|\mathbf{c}_i\|_1, \tag{7.9}$$

where λ is a parameter that needs to be tuned to achieve the desired level of sparsity. Sadly, (7.9) has no closed-form solution, but luckily it is convex, and can be solved efficiently using numerical methods, such as []. In fact, devising better methods to solve ℓ_1-type problems like (7.9) is a prolific active field of research.