Week 9: Dimensionality Reduction

Instructor: Sergey Levine

1 Dimensionality Reduction

In the preceding lectures, we discussed clustering. One view of clustering is that it's a way to summarize a complex real-valued datapoint $\mathbf{x} \in \mathbb{R}^D$ with a single categorical variable $y \in \{1, \dots, K\}$. This could be useful for understanding and visualizing the structure in the data, as well as a preprocessing step for other learning algorithms. For example, we could build a very simple classifier on top of y instead of dealing with all the complexity of $\mathbf{x}$. In this lecture, we'll discuss another way to simplify complex, high-dimensional data for visualization or subsequent (supervised learning), where instead of summarizing the datapoint $\mathbf{x}$ with a categorical variable y, we instead summarize it with a smaller real-valued vector $\mathbf{z}$. For example, $\mathbf{x}$ might represent all of the pixels in an image of a face, or all of the vertices in a 3D scan of a person's body – thousands or millions of dimensions – while $\mathbf{z}$ might consist of only a small number of parameters, such as the person's height and weight, or the direction the face in the image is pointing. Summarizing continuous high-dimensional datapoints with continuous low-dimensional datapoints is called dimensionality reduction. The lecture slides show a few examples of dimensionality reduction problems.

2 Dimensionality Reduction as Feature Learning

One way to look at dimensionality reduction is in terms of features. Recall that for most of the algorithms we discussed, we operate not on the raw input $\mathbf{x}$, but on some features $h(\mathbf{x})$. For example, a linear regression model might make predictions according to

$$y = \mathbf{w} \cdot h(\mathbf{x}),$$

and logistic regression might output probabilities according to

$$p(y = 1) = \frac{1}{1 + \exp(-\mathbf{w} \cdot h(\mathbf{x}))}.$$

So far, we've usually taken the feature function $h(\mathbf{x})$ to be something that maps $\mathbf{x}$ to a *higher* dimensional space – for example by appending a bias feature 1, adding higher-order monomials such as $\mathbf{x}_1^2, \mathbf{x}_2\mathbf{x}_1, \mathbf{x}_2^2$, or adding other functions

of $\mathbf{x}$. Adding more expressive features can reduce bias by allowing the algorithm to learn more complex functions, like higher order polynomials. But what if the problem is not bias but variance? Can we use the feature function to *take away* dimensions from the data, so that the weights $\mathbf{w}$ are lower dimensional and we overfit less? Indeed we can. We could do this manually, or we could devise an automatic method that summarizes the data with a lower-dimensional vector while preserving as much information as possible.

As with all machine learning problems, we need a dataset, a hypothesis space, an objective, and an algorithm. We will be doing unsupervised learning, so our dataset consists of just the inputs $\mathbf{x} \in \mathbb{R}^D$: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. This is convenient because then we can use the same method to reduce dimensionality for supervised learning, visualize data by reducing its dimensionality to 2 or 3, etc. We also need a hypothesis space. There are a few choices, but we'll start with a simple linear projection: we'll say that our goal is to learn a set of basis vectors $\mathbf{u}_1, \ldots, \mathbf{u}_K$ so that the features are given by

$$h(\mathbf{x}) = \begin{bmatrix} \mathbf{u}_1 \cdot \mathbf{x} \\ \mathbf{u}_2 \cdot \mathbf{x} \\ \cdots \\ \mathbf{u}_M \cdot \mathbf{x} \end{bmatrix}.$$

Then we can summarize a D-dimensional vector $\mathbf{x}$ with a K-dimensional feature vector $h(\mathbf{x})$. For convenience, we'll $z_j = \mathbf{u}_j \cdot \mathbf{x}$, so that $h(\mathbf{x}) = [z_1, \ldots, z_K]^T = \mathbf{z}$. Using simple linear projections will provide us with an extremely efficient algorithm, if we are a little bit careful in how we choose $\mathbf{u}_1, \ldots, \mathbf{u}_K$.

First, note that if we scale $\mathbf{u}_j$ by some constant c, then z_j scales by the same constant. This is not very interesting – we might as well pick one scale for $\mathbf{u}_j$ and stick with it, since changing the scale doesn't change the features in an interesting way. So we'll just force $||\mathbf{u}_j|| = 1$. This also means that $\mathbf{u}_j^T \mathbf{u}_j = 1$. The second constraint we'll impose is a bit nuanced, but it also makes sense: we'll force $\mathbf{u}_j^T \mathbf{u}_i = 0$ for all $i \neq j$. Why? Well, imagine that $\mathbf{u}_j^T \mathbf{u}_i \neq 0$, we know that we can decompose $\mathbf{u}_i$ into two vectors, such that $\mathbf{u}_i = \mathbf{u}_i^0 + \mathbf{u}_i^1$, where $\mathbf{u}_i^0$ is the portion of $\mathbf{u}_i$ that is orthogonal to $\mathbf{u}_j$, such that $\mathbf{u}_j^T \mathbf{u}_i^0 = 0$. Then the weight z_i will have a portion that is linearly depend on z_j – that is, part of z_i will contain the same information as z_j. Specifically, we'll have $z_i = \mathbf{u}_i^1 \cdot \mathbf{x} + \mathbf{u}_i^0 \cdot \mathbf{u}_j z_j$. Notice that the second part doesn't really contribute any new information – it doesn't even depend on the input $\mathbf{x}$! So we might as well force $\mathbf{u}_i^T \mathbf{u}_j = 0$ and keep all of the basis vectors orthogonal. This will also make it very convenient to obtain the best basis vectors later.

So, to summarize, our hypothesis class consists of K vectors $\mathbf{u}_1, \cdot, \mathbf{u}_K$, such that $\mathbf{u}_i^T \mathbf{u}_i = 1$ and $\mathbf{u}_i^T \mathbf{u}_j = 1$ when $i \neq j$. We can arrange these vectors into a matrix

$$\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_K].$$

This matrix orthonormal: the columns are all orthogonal and unit length. Our features are given by

$$h(\mathbf{x}) = \mathbf{U}^T \mathbf{x}$$

In the sequel, we'll change this definition a little bit. Instead of setting $h(\mathbf{x}) = \mathbf{U}^T\mathbf{x}$, we'll instead set

$$h(\mathbf{x}) = \mathbf{U}^T(\mathbf{x} - \mu_0),$$

where $\mu_0 = \frac{1}{N}\sum_{i=1}^N \mathbf{x}_i$ (the empirical mean of the data). The reason for this seems a little mysterious at first, but it will become a bit more apparent later. Crucially, this change adds only a constant to the features, so functionally they will behave mostly the same way.

3 Probabilistic Model of Dimensionality Reduction

Now that we have a hypothesis class, we'll need an objective. As with clustering, a good unsupervised objective is the log likelihood:

$$\log p(\mathcal{D}) = \sum_{i=1}^N \log p(\mathbf{x}).$$

We need to pick a distribution $p(\mathbf{x})$. Since $\mathbf{x} \in \mathbb{R}^D$, a good choice would be a Gaussian, just like in the case of linear regression, although now we are modeling $\mathbf{x}$, not y. In this Gaussian, we'll set the mean to be $\mu = \mathbf{U}\mathbf{z} + \mu_0$. We'll use unit variance. That way, the log probability is given by

$$\log p(\mathcal{D}) = \sum_{i=1}^N \sum_{j=1}^D -\frac{1}{2}((\mathbf{U}\mathbf{z}_i)_j - (x_{i,j} - \mu_{0,j}))^2 + \text{const.}$$

We can write this in vector notation as

$$\log p(\mathcal{D}) = \sum_{i=1}^N -\frac{1}{2}||\mathbf{U}\mathbf{z}_i - (\mathbf{x}_i - \mu_0)||^2 + \text{const.}$$

Why did we choose $\mu = \mathbf{U}\mathbf{z} + \mu_0$? Well, if we maximize the log-likelihood with respect to $\mathbf{z}_i$ by taking the derivative and setting it to zero, we recover the familiar normal equations, as is typically the case in linear regression:

$$\mathbf{z}_i = (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T(\mathbf{x} - \mu_0).$$

However, we know that $\mathbf{U}$ is orthonormal, and therefore $\mathbf{U}^T\mathbf{U} = \mathbf{I}_{K\times K}$, so we simply have

$$\mathbf{z}_i = \mathbf{U}^T(\mathbf{x} - \mu_0).$$

So we see that $\mu = \mathbf{U}\mathbf{z} + \mu_0$ is the optimal setting for the mean of this model for the hypothesis space that we defined previously, and this also illustrates why we chose $\mathbf{U}$ to be orthonormal. How convenient!

Now we have a hypothesis space and an objective, so all that remains is to devise an algorithm that can find the best orthonormal basis matrix $\mathbf{U}$ that will maximize the log-likelihood of the dataset.

4 Optimizing the Basis: Principal Component Analysis

Now we are ready to define the optimization problem and devise an algorithm. Since the log-likelihood is the negative of the norm of $\mathbf{U}\mathbf{z}_i - (\mathbf{x}_i - \mu_0)$, we can conveniently write the optimization over $\mathbf{U}$ as a minimization:

$$\mathbf{U} \leftarrow \arg\min_{\mathbf{U}} \sum_{i=1}^{N} \frac{1}{2N} ||\mathbf{U}\mathbf{z}_i - \bar{\mathbf{x}}_i||^2 \text{ such that } \mathbf{u}_i^T \mathbf{u}_i = 1 \text{ and } \mathbf{u}_i^T \mathbf{u}_j = 0 \ \forall i \neq j$$

Where we use $\bar{\mathbf{x}}_i = \mathbf{x}_i - \mu_0$ and added a division by N (this will be convenient later). We can rearrange this problem since we know that $\mathbf{z}_i = \mathbf{U}^T \bar{\mathbf{x}}$:

$$\begin{aligned}
\sum_{i=1}^{N} \frac{1}{N2} ||\mathbf{U}\mathbf{z}_i - \bar{\mathbf{x}}_i||^2 &= \sum_{i=1}^{N} \frac{1}{N2} ||\mathbf{U}\mathbf{U}^T \bar{\mathbf{x}}_i - \bar{\mathbf{x}}_i||^2 \\
&= \sum_{i=1}^{N} \frac{1}{N2} \bar{\mathbf{x}}_i^T \mathbf{U}\mathbf{U}^T \mathbf{U}\mathbf{U}^T \bar{\mathbf{x}}_i - \frac{1}{N} \bar{\mathbf{x}}_i^T \mathbf{U}\mathbf{U}^T \bar{\mathbf{x}}_i + \frac{1}{2N} \bar{\mathbf{x}}_i^T \bar{\mathbf{x}}_i.
\end{aligned}$$

Using the fact that $\mathbf{U}$ is orthonormal to simplify the initial sum, we get

$$\begin{aligned}
\sum_{i=1}^{N} \frac{1}{2N} ||\mathbf{U}\mathbf{z}_i - \bar{\mathbf{x}}_i||^2 &= \sum_{i=1}^{N} \frac{1}{2N} \bar{\mathbf{x}}_i^T \mathbf{U}\mathbf{U}^T \mathbf{U}\mathbf{U}^T \bar{\mathbf{x}}_i - \frac{1}{N} \bar{\mathbf{x}}_i^T \mathbf{U}\mathbf{U}^T \bar{\mathbf{x}}_i + \frac{1}{2N} \bar{\mathbf{x}}_i^T \bar{\mathbf{x}}_i \\
&= \sum_{i=1}^{N} \frac{1}{2N} \bar{\mathbf{x}}_i^T \mathbf{U}\mathbf{U}^T \bar{\mathbf{x}}_i - \frac{1}{N} \bar{\mathbf{x}}_i^T \mathbf{U}\mathbf{U}^T \bar{\mathbf{x}}_i + \frac{1}{2N} \bar{\mathbf{x}}_i^T \bar{\mathbf{x}}_i \\
&= \sum_{i=1}^{N} -\frac{1}{2N} \bar{\mathbf{x}}_i^T \mathbf{U}\mathbf{U}^T \bar{\mathbf{x}}_i + \frac{1}{2N} \bar{\mathbf{x}}_i^T \bar{\mathbf{x}}_i
\end{aligned}$$

Now, if we consider the first term in the sum, we can write it out like this:

$$\bar{\mathbf{x}}_i^T \mathbf{U}\mathbf{U}^T \bar{\mathbf{x}}_i = \sum_{k=1}^{K} \mathbf{u}_k^T \bar{\mathbf{x}}_i \bar{\mathbf{x}}_i^T \mathbf{u}_k,$$

where $\mathbf{u}_k$ is the k^{th} column of $\mathbf{U}$. Checking this is left as an exercise, but it can be checked simply by writing out the formula for vector-matrix multiplication

as a summation. Now, we can move the sum inside the product to get:

$$\begin{aligned}\sum_{i=1}^{N}\frac{1}{2N}||\mathbf{U}\mathbf{z}_i-\bar{\mathbf{x}}_i||^2 &= \sum_{i=1}^{N}-\frac{1}{2N}\bar{\mathbf{x}}_i^T\mathbf{U}\mathbf{U}^T\bar{\mathbf{x}}_i+\frac{1}{2N}\bar{\mathbf{x}}_i^T\bar{\mathbf{x}}_i\\ &= \sum_{i=1}^{N}-\frac{1}{2N}\sum_{k=1}^{K}\mathbf{u}_k^T\bar{\mathbf{x}}_i\bar{\mathbf{x}}_i^T\mathbf{u}_k+\frac{1}{2N}\bar{\mathbf{x}}_i^T\bar{\mathbf{x}}_i\\ &= -\frac{1}{2N}\sum_{k=1}^{K}\mathbf{u}_k^T\sum_{i=1}^{N}[\bar{\mathbf{x}}_i\bar{\mathbf{x}}_i^T]\mathbf{u}_k+\sum_{i=1}^{N}\frac{1}{2N}\bar{\mathbf{x}}_i^T\bar{\mathbf{x}}_i\end{aligned}$$

Note that

$$\frac{1}{N}\sum_{i=1}^{N}\bar{\mathbf{x}}_i\bar{\mathbf{x}}_i^T=\frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i-\mu_0)(\mathbf{x}_i-\mu_0)^T=\Sigma,$$

where Σ is the empirical covariance of the data that we saw before when we discussed multivariate Gaussians! So our objective becomes

$$\sum_{i=1}^{N}\frac{1}{2N}||\mathbf{U}\mathbf{z}_i-\bar{\mathbf{x}}_i||^2=-\frac{1}{2}\sum_{k=1}^{K}\mathbf{u}_k^T\Sigma\mathbf{u}_k+\text{const},$$

where the constant doesn't depend on $\mathbf{U}$. Going back to our optimization, we therefore have

$$\mathbf{U}\leftarrow\arg\min_{\mathbf{U}}-\frac{1}{2}\sum_{k=1}^{K}\mathbf{u}_k^T\Sigma\mathbf{u}_k \text{ such that } \mathbf{u}_i^T\mathbf{u}_i=1 \text{ and } \mathbf{u}_i^T\mathbf{u}_j=0\ \ \forall i\neq j.$$

Now we're ready to take the derivative and set it to zero, but we have to take care of the constraints. Recall that we can handle constraints using Lagrange multipliers! First, let's see what happens in the simple case where $K=1$ and $\mathbf{U}$ has just one column. Add in the Lagrange multipliers (we multiply by $\frac{1}{2}$ for convenience):

$$\mathcal{L}(\mathbf{u},\lambda)=-\frac{1}{2}\mathbf{u}^T\Sigma\mathbf{u}+\frac{1}{2}\lambda(\mathbf{u}^T\mathbf{u}-1).$$

Now take the derivative:

$$\frac{d\mathcal{L}}{d\mathbf{u}}=-\Sigma\mathbf{u}+\lambda\mathbf{u}=0\Rightarrow\Sigma\mathbf{u}=\lambda\mathbf{u}$$

That's interesting! It's not immediately straightforward to solve for $\mathbf{u}$, until we recognize that this is exactly the definition of eigenvectors and eigenvalues! So $\mathbf{u}$ must be an eigenvector of Σ, and λ must be an eigenvalue. All eigenvectors $\mathbf{u}$ will satisfy the constraint, so if we substitute the solution into the objective, we get

$$\min_{\mathbf{u}}-\frac{1}{2}\mathbf{u}^T\Sigma\mathbf{u}=\min_{\mathbf{u}}-\frac{1}{2}\mathbf{u}^T\mathbf{u}\lambda=\min_{\mathbf{u}}-\frac{1}{2}\lambda,$$

where the last step follows from the fact that $\mathbf{u}^T\mathbf{u} = 1$. So our goal is simply to maximize the eigenvalue that corresponds to the eignevector $\mathbf{u}$! Therefore, we have only one basis vector, it should be the eigenvector of Σ that corresponds to the largest eigenvalue.

What happens if we have more than one basis vector ($K > 1$)? Well, we could repeat the same exercise with Lagrange multipliers, but we could observe that, were it not for the orthogonality constraints $\mathbf{u}_i^T\mathbf{u}_j = 0$ for $i \neq j$, the rest of the Lagrangian factorizes additively (that is, all $\mathbf{u}_k$ vectors are independent), so we always have

$$\Sigma\mathbf{u}_k = \lambda_k\mathbf{u}_k,$$

and we always want to maximize the corresponding eigenvalues λ_k, since we have

$$-\frac{1}{2}\sum_{k=1}^{K}\mathbf{u}_k^T\Sigma\mathbf{u}_k = -\frac{1}{2}\sum_{k=1}^{K}\lambda_k.$$

Therefore, since all $\mathbf{u}_k$ vectors must be orthogonal, they must all be *different* eigenvectors, and we should pick the ones that correspond to the K *largest* eigenvalues. We therefore recover the simple algorithm for obtaining the optimal basis to maximize $\log p(\mathcal{D})$: compute an eigenvalue decomposition of the empirical covariance

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \mu_0)(\mathbf{x}_i - \mu_0)^T,$$

and then populate the columns of $\mathbf{U}$ with the eigenvectors of Σ corresponding to the K largest eigenvalues. This is called principal component analysis.