COMS 4721: Machine Learning for Data Science
Lecture 19, 4/6/2017

Prof. John Paisley

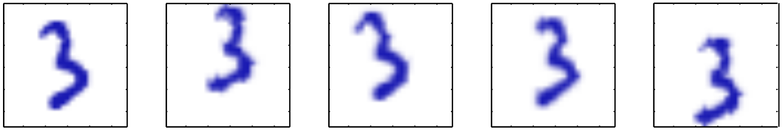
Department of Electrical Engineering
& Data Science Institute

Columbia University

Principal Component Analysis

DIMENSIONALITY REDUCTION

We're given data $x_1, \ldots, x_n$, where $x \in \mathbb{R}^d$. This data is often high-dimensional, but the "information" doesn't use the full d dimensions.



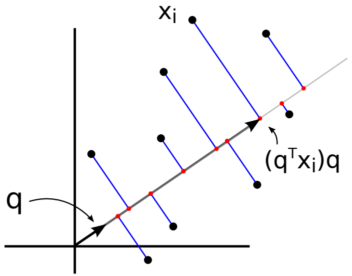
For example, we could represent the above images with three numbers since they have three degrees of freedom. Two for shifts and a third for rotation.

Principal component analysis can be thought of as a way of automatically mapping data x_i into some new low-dimensional coordinate system.

- It capture most of the information in the data in a few dimensions
- Extensions allow us to handle missing data, and "unwrap" the data.

Principal component analysis

Example: How can we approximate this data using a unit-length vector q?



q is a unit-length vector, $q^T q = 1$.

Red dot: The length, $q^T x_i$, to the axis after projecting x onto the line defined by q.

The vector $(q^T x_i)q$ takes q and stretches it to the corresponding red dot.

So what's a good q? How about minimizing the squared approximation error,

$$q = \arg\min_q \sum_{i=1}^{n} \|x_i - qq^T x_i\|^2 \qquad \text{subject to} \quad q^T q = 1$$

$qq^T x_i = (q^T x_i)q$: The approximation of x_i by stretching q to the "red dot."

PCA : THE FIRST PRINCIPAL COMPONENT

This is related to the problem of finding the largest eigenvalue,

$$\begin{aligned} q &= \arg\min_q \sum_{i=1}^n \|x_i - qq^T x_i\|^2 \qquad \text{s.t.} \quad q^T q = 1 \\ &= \arg\min_q \sum_{i=1}^n x_i^T x_i - q^T \underbrace{\left(\sum_{i=1}^n x_i x_i^T\right)}_{=XX^T} q \end{aligned}$$

We've defined $X = [x_1, \ldots, x_n]$. Since the first term doesn't depend on q and we have a negative sign in front of the second term, equivalently we solve

$$q = \arg\max_q \; q^T(XX^T)q \qquad \text{subject to} \quad q^T q = 1$$

This is the eigendecomposition problem:

- q is the first eigenvector of XX^T
- $\lambda = q^T(XX^T)q$ is the first eigenvalue

PCA: General

The general form of PCA considers K eigenvectors,

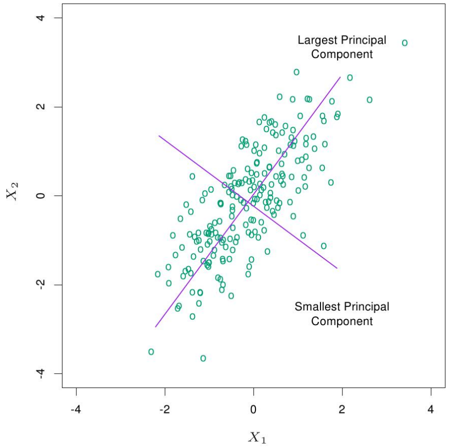
$$\begin{aligned} q &= \arg\min_q \sum_{i=1}^n \| x_i - \underbrace{\sum_{k=1}^K (x_i^T q_k) q_k}_{\text{approximates } x} \|^2 \qquad \text{s.t. } q_k^T q_{k'} = \begin{cases} 1, & k = k' \\ 0, & k \neq k' \end{cases} \\ &= \arg\min_q \sum_{i=1}^n x_i^T x_i - \sum_{k=1}^K q_k^T \underbrace{\left(\sum_{i=1}^n x_i x_i^T \right)}_{= XX^T} q_k \end{aligned}$$

The vectors in $Q = [q_1, \dots, q_K]$ give us a K dimensional subspace with which to represent the data:

$$x_{\text{proj}} = \begin{bmatrix} q_1^T x \\ \vdots \\ q_K^T x \end{bmatrix}, \qquad x \approx \sum_{k=1}^K (q_k^T x) q_k = Q x_{\text{proj}}$$

The eigenvectors of (XX^T) can be learned using built-in software.

EIGENVALUES, EIGENVECTORS AND THE SVD



An equivalent formulation of the problem is to find (λ, q) such that

$$(XX^T)q = \lambda q$$

Since (XX^T) is a PSD matrix, there are $r \leq \min\{d, n\}$ pairs,

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0,$$

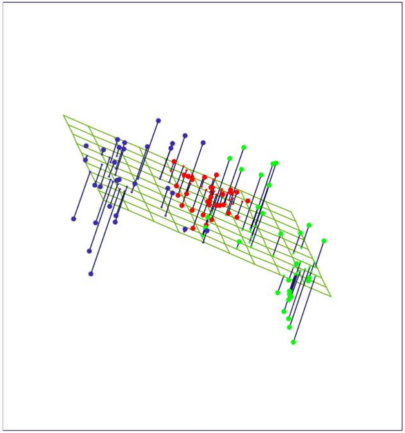
$$q_k^T q_k = 1, \quad q_k^T q_{k'} = 0$$

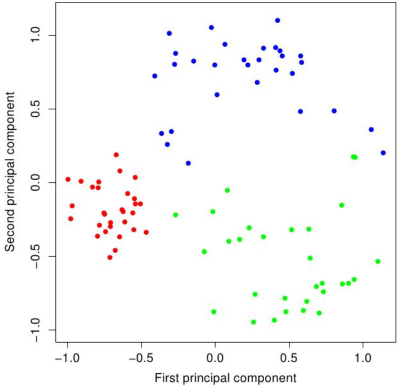
Why is (XX^T) PSD? Using the SVD, $X = USV^T$, we have that

$$(XX^T) = US^2U^T \quad \Rightarrow \quad Q = U, \quad \lambda_i = (S^2)_{ii} \geq 0$$

Preprocessing: Usually we first subtract off the mean of each dimension of x.

PCA: Example of projecting from $\mathbb{R}^3$ to $\mathbb{R}^2$





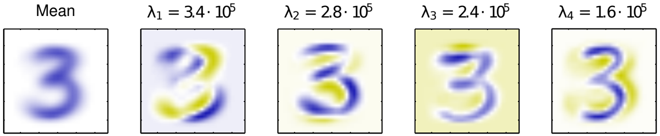
For this data, most information (structure in the data) can be captured in $\mathbb{R}^2$.

(left) The original data in $\mathbb{R}^3$. The hyperplane is defined by q_1 and q_2.

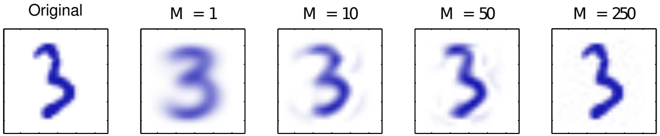
(right) The new coordinates for the data: $x_i \rightarrow x_i^{proj} = \begin{bmatrix} x_i^T q_1 \\ x_i^T q_2 \end{bmatrix}$.

EXAMPLE: DIGITS

Data: 16×16 images of handwritten 3's (as vectors in $\mathbb{R}^{256}$)



Above: The first four eigenvectors q and their eigenvalues λ.



Above: Reconstructing a 3 using the first $M-1$ eigenvectors plus the mean, and approximation

$$x \approx \text{mean} + \sum_{k=1}^{M-1} (x^T q_k) q_k$$

Probabilistic PCA

PCA AND THE SVD

We've discussed how any matrix X has a singular value decomposition,

$$X = USV^T, \quad U^T U = I, \quad V^T V = I$$

and S is a diagonal matrix with non-negative entries.

Therefore,

$$XX^T = US^2 U^T \quad \Leftrightarrow \quad (XX^T)U = US^2$$

U is a matrix of eigenvectors, and S^2 is a diagonal matrix of eigenvalues.

A modeling approach to PCA

Using the SVD perspective of PCA, we can also derive a probabilistic model for the problem and use the EM algorithm to learn it.

This model will have the advantages of:

- Handling the problem of missing data
- Allowing us to learn additional parameters such as noise
- Provide a framework that could be extended to more complex models
- Gives distributions used to characterize uncertainty in predictions
- etc.

Probabilistic PCA

In effect, this is a new matrix factorization model.

- With the SVD, we had $X = USV^T$.
- We now approximate $X \approx WZ$, where
  - W is a $d \times K$ matrix. In different settings this is called a “factor loadings” matrix, or a “dictionary.” It’s like the eigenvectors, but no orthonormality.
  - The ith column of Z is called $z_i \in \mathbb{R}^K$. Think of it as a low-dimensional representation of x_i.

The generative process of Probabilistic PCA is

$$x_i \sim N(Wz_i, \sigma^2 I), \qquad z_i \sim N(0, I).$$

In this case, we don’t know W or any of the z_i.

THE LIKELIHOOD

Maximum likelihood

Our goal is to find the maximum likelihood solution of the matrix W under the marginal distribution, i.e., with the z_i vectors integrated out,

$$W_{\text{ML}} = \arg\max_W \; \ln p(x_1, \dots, x_n | W) = \arg\max_W \; \sum_{i=1}^{n} \ln p(x_i | W).$$

This is intractable because $p(x_i|W) = N(x_i|0, \sigma^2 I + WW^T)$,

$$N(x_i|0, \sigma^2 I + WW^T) = \frac{1}{(2\pi)^{\frac{d}{2}} |\sigma^2 I + WW^T|^{\frac{1}{2}}} \, e^{-\frac{1}{2} x^T (\sigma^2 I + WW^T)^{-1} x}$$

We can set up an EM algorithm that uses the vectors $z_1, \dots, z_n$.

EM for Probabilistic PCA

Setup

The marginal log likelihood can be expressed using EM as

$$\begin{aligned}\sum_{i=1}^{n} \ln \int p(x_i, z_i|W)\, dz_i \quad = & \quad \sum_{i=1}^{n} \int q(z_i) \ln \frac{p(x_i, z_i|W)}{q(z_i)}\, dz_i \qquad \leftarrow \quad \mathcal{L} \\ & + \sum_{i=1}^{n} \int q(z_i) \ln \frac{q(z_i)}{p(z_i|x_i, W)}\, dz_i \qquad \leftarrow \quad \text{KL}\end{aligned}$$

EM Algorithm: Remember that EM has two iterated steps

1. Set $q(z_i) = p(z_i|x_i, W)$ for each i (making $\text{KL} = 0$) and calculate $\mathcal{L}$
2. Maximize $\mathcal{L}$ with respect to W

Again, for this to work well we need that

- we can calculate the posterior distribution $p(z_i|x_i, W)$, and
- maximizing $\mathcal{L}$ is easy, i.e., we update W using a simple equation

EM for Probabilistic PCA

Given: Data $x_{1:n}$, $x_i \in \mathbb{R}^d$ and model $x_i \sim N(Wz_i, \sigma^2)$, $z_i \sim N(0, I)$, $z \in \mathbb{R}^K$

Output: Point estimate of W and posterior distribution on each z_i

E-Step: Set each $q(z_i) = p(z_i|x_i, W) = N(z_i|\mu_i, \Sigma_i)$ where

$$\Sigma_i = (I + W^T W/\sigma^2)^{-1}, \quad \mu_i = \Sigma_i W^T x_i/\sigma^2$$

M-Step: Update W by maximizing the objective $\mathcal{L}$ from the E-step

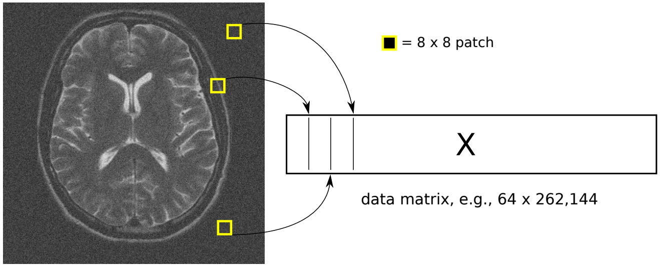
$$W = \left[\sum_{i=1}^n x_i \mu_i^T\right] \left[\sigma^2 I + \sum_{i=1}^n (\mu_i \mu_i^T + \Sigma_i)\right]^{-1}$$

Iterate E and M steps until increase in $\sum_{i=1}^n \ln p(x_i|W)$ is "small."

Comment:

- The probabilistic framework gives a way to learn K and σ^2 as well.

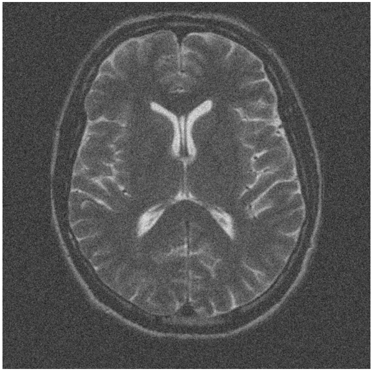
EXAMPLE: IMAGE PROCESSING

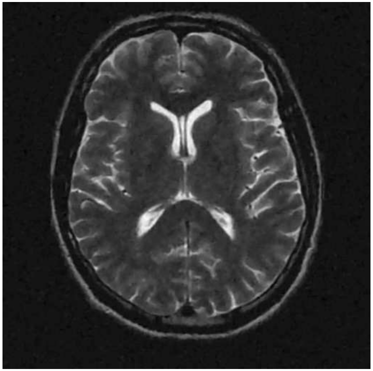


For image problems such as denoising or inpainting (missing data)

- Extract overlapping patches (e.g., 8×8) and vectorize to construct X
- Model with a factor model such as Probabilistic PCA
- Approximate $x_i \approx W\mu_i$, where μ_i is the posterior mean of z_i
- Reconstruct the image by replacing x_i with $W\mu_i$ (and averaging)

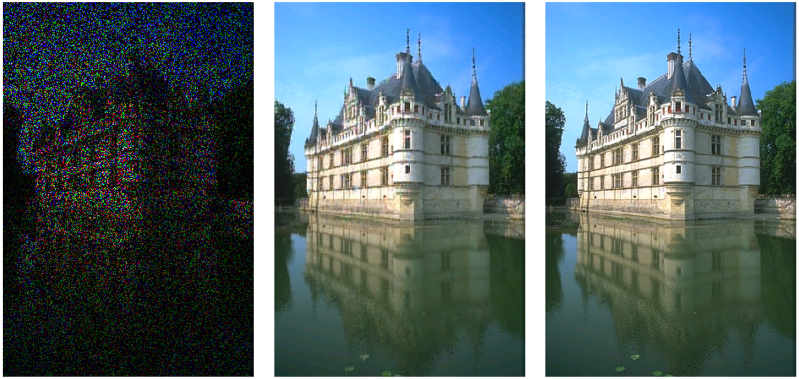
EXAMPLE: DENOISING





Noisy image on left, denoised image on right. The noise variance parameter σ^2 was learned for this example.

EXAMPLE: MISSING DATA



Another somewhat extreme example:

- Image is $480 \times 320 \times 3$ (RGB dimension)
- Throw away 80% at random
- (left) Missing data, (middle) reconstruction, (right) original image

Kernel PCA

Kernel PCA

We've seen how we can take an algorithm that uses dot products, $x^T x$, and generalize with a nonlinear kernel. This generalization can be made to PCA.

Recall: With PCA we find the eigenvectors of the matrix $\sum_{i=1}^n x_i x_i^T = XX^T$.

- Let $\phi(x)$ be a feature mapping from $\mathbb{R}^d$ to $\mathbb{R}^D$, where $D \gg d$
- We want to solve the eigendecomposition

$$\left[\sum_{i=1}^{n} \phi(x_i)\phi(x_i)^T\right] q_k = \lambda_k q_k$$

  without having to work in the higher dimensional space.
- That is, how can we do PCA without explicitly using $\phi(\cdot)$ and q?

Kernel PCA

Notice that we can reorganize the operations of the eigendecomposition

$$\sum_{i=1}^{n} \phi(x_i) \underbrace{\left(\phi(x_i)^T q_k\right) / \lambda_k}_{= a_{ki}} = q_k$$

That is, the eigenvector $q_k = \sum_{i=1}^{n} a_{ki}\phi(x_i)$ for some vector $\boldsymbol{a}_k \in \mathbb{R}^n$.

The trick is that instead of learning q_k, we'll learn $\boldsymbol{a}_k$.

Plug this equation for q_k back into the first equation:

$$\sum_{i=1}^{N} \phi(x_i) \sum_{j=1}^{n} a_{kj} \underbrace{\phi(x_i)^T \phi(x_j)}_{= K(x_i, x_j)} = \lambda_k \sum_{i=1}^{n} a_{ki}\phi(x_i)$$

and multiply both sides by $\phi(x_l)^T$ for each $l \in \{1, \ldots, n\}$.

Kernel PCA

When we multiply the following by $\phi(x_l)^T$ for $l = 1 \ldots, n$:

$$\sum_{i=1}^{N} \phi(x_i) \sum_{j=1}^{n} a_{kj} \underbrace{\phi(x_i)^T \phi(x_j)}_{= K(x_i, x_j)} = \lambda_k \sum_{i=1}^{n} a_{ki} \phi(x_i)$$

we get a new set of linear equations

$$K^2 \boldsymbol{a}_k = \lambda_k K \boldsymbol{a}_k \quad \Longleftrightarrow \quad K \boldsymbol{a}_k = \lambda_k \boldsymbol{a}_k$$

where K is the $n \times n$ *kernel matrix* constructed on the data.

Because K is guaranteed to be PSD because it is a matrix of dot-products, the LHS and RHS above share a solution for $(\lambda_k, \boldsymbol{a}_k)$.

Now perform "regular" PCA, but on the kernel matrix K instead of the data matrix XX^T. We summarize the algorithm on the following slide.

KERNEL PCA ALGORITHM

Kernel PCA

Given: Data $x_1, \ldots, x_n$, $x \in \mathbb{R}^d$, and a kernel function $K(x_i, x_j)$.

Construct: The kernel matrix on the data, e.g., $K_{ij} = b \exp\left\{-\frac{\|x_i - x_j\|^2}{c}\right\}$.

Solve: The eigendecomposition

$$K\boldsymbol{a}_k = \lambda_k \boldsymbol{a}_k$$

for the first $r \ll n$ eigenvector/eigenvalue pairs $(\lambda_1, \boldsymbol{a}_1), \ldots, (\lambda_r, \boldsymbol{a}_r)$.

Output: A new coordinate system for x_i by (implicitly) mapping $\phi(x_i)$ and then projecting $q_k^T \phi(x_i)$

$$x_i \quad \xrightarrow{\text{projection}} \quad \begin{bmatrix} \lambda_1 a_{1i} \\ \vdots \\ \lambda_r a_{ri} \end{bmatrix}$$

where a_{ki} is the ith dimension of the kth eigenvector $\boldsymbol{a}_k$.

Kernel PCA and new data

Q: How do we handle new data, x_0? Before, we could take the eigenvectors q_k and project $x_0^T q_k$, but $\boldsymbol{a}_k$ is different here.

A: Recall the relationship of $\boldsymbol{a}_k$ to q_k in kernel PCA is

$$q_k = \sum_{i=1}^{n} a_{ki}\phi(x_i).$$

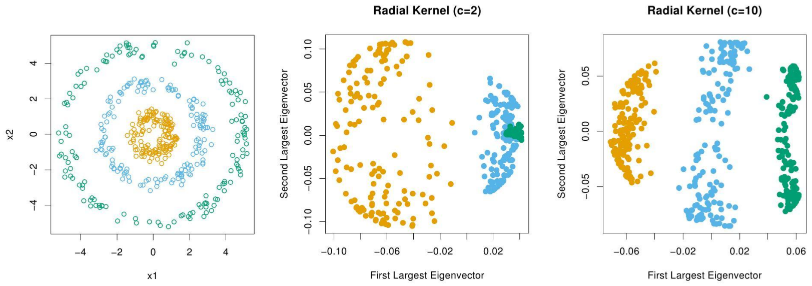
We used the "kernel trick" to avoid working with or even defining $\phi(x_i)$.

As with regular PCA, after mapping x_0 we want to project onto eigenvectors

$$x_0 \quad \xrightarrow{\text{projection}} \quad \left[\begin{array}{c} \phi(x_0)^T q_1 \\ \vdots \\ \phi(x_0)^T q_r \end{array}\right]$$

Plugging in for q_k: $\quad \phi(x_0)^T q_k = \sum_{i=1}^{n} a_{ki}\phi(x_0)^T\phi(x_i) = \sum_{i=1}^{n} a_{ki}K(x_0, x_i).$

EXAMPLE RESULTS



An example of kernel PCA using the Gaussian kernel.

(left) Original data, colored for reference (but may be classes)

(middle) New coordinates using kernel width $c = 2$

(right) New coordinates using kernel width $c = 10$

Terminology: What we are doing is closely related to “spectral clustering” and can be considered an instance of “manifold learning.”