Linear Dimensionality Reduction

Practical Machine Learning (CS294-34)

September 24, 2009

Percy Liang

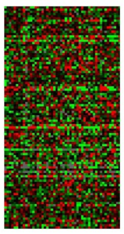
Lots of high-dimensional data...

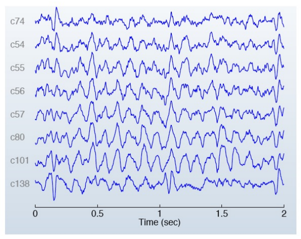
face images

Zambian President Levy Mwanawasa has won a second term in office in an election his challenger Michael Sata accused him of rigging, official results showed on Monday.

According to media reports, a pair of hackers said on Saturday that the Firefox Web browser, commonly perceived as the safer and more customizable alternative to market leader Internet Explorer, is critically flawed. A presentation on the flaw was shown during the ToorCon hacker conference in San Diego.

documents


gene expression data



MEG readings

In many real applications, we are confronted with various types of high-dimensional data. The goal of dimensionality reduction is to convert this data into a lower dimensional representation more amenable to visualization or further processing by machine learning algorithms.

Motivation and context

Why do dimensionality reduction?

- Computational: compress data $\Rightarrow$ time/space efficiency
- Statistical: fewer dimensions $\Rightarrow$ better generalization
- Visualization: understand structure of data
- Anomaly detection: describe normal data, detect outliers

Dimensionality reduction in this course:

- Linear methods (this week)
- Clustering (last week)
- Feature selection (next week)
- Nonlinear methods (later)

There are several reasons one might want to do dimensionality reduction. We have already seen one way of effectively reducing the dimensionality of data (clustering), and we will see others in future lectures.

Types of problems

- Prediction $\mathbf{x} \to \mathbf{y}$: classification, regression
  Applications: face recognition, gene expression prediction
  Techniques: kNN, SVM, least squares (+ dimensionality reduction preprocessing)

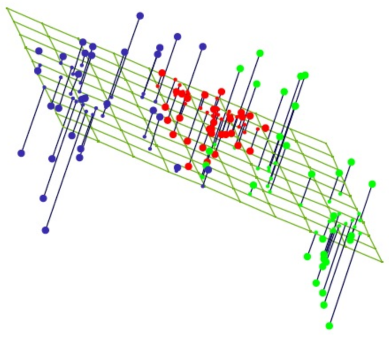
- Structure discovery $\mathbf{x} \to \mathbf{z}$: find an alternative representation $\mathbf{z}$ of data $\mathbf{x}$
  Applications: visualization
  Techniques: clustering, linear dimensionality reduction

- Density estimation $p(\mathbf{x})$: model the data
  Applications: anomaly detection, language modeling
  Techniques: clustering, linear dimensionality reduction

Here are three typical classes of problems where dimensionality reduction can be beneficial.

Basic idea of linear dimensionality reduction

Represent each face as a high-dimensional vector $\mathbf{x} \in \mathbb{R}^{361}$



$$\begin{tikzcd} \mathbf{x} \in \mathbb{R}^{361} \arrow[d, "\mathbf{z} = \mathbf{U}^\top \mathbf{x}"] \\ \mathbf{z} \in \mathbb{R}^{10} \end{tikzcd}$$

How do we choose $\mathbf{U}$?

All of the methods we will present fall under this framework. A high-dimensional data point (for example, a face image) is mapped via a linear projection into a lower-dimensional point. An important question to bear in mind is whether a linear projection even makes sense. There are several ways to optimize $\mathbf{U}$ based on the nature of the data.

Outline

- Principal component analysis (PCA)
  - Basic principles
  - Case studies
  - Kernel PCA
  - Probabilistic PCA
- Canonical correlation analysis (CCA)
- Fisher discriminant analysis (FDA)
- Summary

Roadmap

- Principal component analysis (PCA)
  - Basic principles
  - Case studies
  - Kernel PCA
  - Probabilistic PCA
- Canonical correlation analysis (CCA)
- Fisher discriminant analysis (FDA)
- Summary

Dimensionality reduction setup

Given n data points in d dimensions: $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$

$$\mathbf{X} = \begin{pmatrix} | & & | \\ \mathbf{x}_1 & \cdots\cdots & \mathbf{x}_n \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times n}$$

Want to reduce dimensionality from d to k

Choose k directions $\mathbf{u}_1, \dots, \mathbf{u}_k$

$$\mathbf{U} = \begin{pmatrix} | & & | \\ \mathbf{u}_1 & \cdot\cdot & \mathbf{u}_k \\ | & & | \end{pmatrix} \in \mathbb{R}^{d \times k}$$

For each $\mathbf{u}_j$, compute “similarity” $z_j = \mathbf{u}_j^\top \mathbf{x}$

Project $\mathbf{x}$ down to $\mathbf{z} = (z_1, \dots, z_k)^\top = \mathbf{U}^\top \mathbf{x}$

How to choose $\mathbf{U}$?

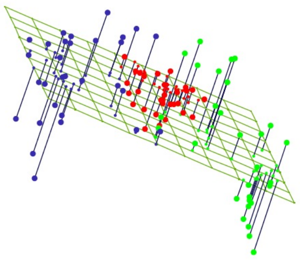
PCA objective 1: reconstruction error

$\mathbf{U}$ serves two functions:

- Encode: $\mathbf{z} = \mathbf{U}^\top \mathbf{x}$, $\quad z_j = \mathbf{u}_j^\top \mathbf{x}$
- Decode: $\tilde{\mathbf{x}} = \mathbf{U}\mathbf{z} = \sum_{j=1}^k z_j \mathbf{u}_j$

Want reconstruction error $\|\mathbf{x} - \tilde{\mathbf{x}}\|$ to be small

Objective: minimize total squared reconstruction error



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

There are two perspectives on PCA. The first one is based on encoding data in a low-dimensional representation so that we can reconstruct the original as well as possible.

PCA objective 2: projected variance

Empirical distribution: uniform over $\mathbf{x}_1, \dots, \mathbf{x}_n$

Expectation (think sum over data points):

$$\hat{\mathbb{E}}[f(\mathbf{x})] = \frac{1}{n} \sum_{i=1}^n f(\mathbf{x}_i)$$

Variance (think sum of squares if centered):

$$\widehat{\mathrm{var}}[f(\mathbf{x})] + (\hat{\mathbb{E}}[f(\mathbf{x})])^2 = \hat{\mathbb{E}}[f(\mathbf{x})^2] = \frac{1}{n} \sum_{i=1}^n f(\mathbf{x}_i)^2$$

Assume data is centered: $\hat{\mathbb{E}}[\mathbf{x}] = 0$ (what's $\hat{\mathbb{E}}[\mathbf{U}^\top \mathbf{x}]$?)

Objective: maximize variance of projected data

$$\max_{\mathbf{U} \in \mathbb{R}^{d \times k}, \mathbf{U}^\top \mathbf{U} = I} \hat{\mathbb{E}}[\|\mathbf{U}^\top \mathbf{x}\|^2]$$

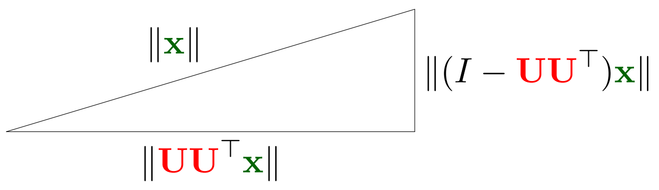
The second viewpoint is that we want to find projections that capture (explain) as much variance in data as possible. Note that we require the column vectors in $\mathbf{U}$ to be orthonormal: $\mathbf{U}^\top \mathbf{U} = I_{k \times k}$ to avoid degenerate solutions of ∞. To talk about variance, we need to talk about a random variable. The random variable here is a data point $\mathbf{x}$, which we assume to be drawn from the uniform distribution over the n data points. Unless otherwise specified, we will always assume the data is centered at zero (can be easily achieved by subtracting out the mean).

Equivalence in two objectives

Key intuition:

$$\underbrace{\text{variance of data}}_{\text{fixed}} = \underbrace{\text{captured variance}}_{\text{want large}} + \underbrace{\text{reconstruction error}}_{\text{want small}}$$

Pythagorean decomposition: $\mathbf{x} = \mathbf{U}\mathbf{U}^\top \mathbf{x} + (I - \mathbf{U}\mathbf{U}^\top)\mathbf{x}$


$\|\mathbf{x}\|$
$\|(I - \mathbf{U}\mathbf{U}^\top)\mathbf{x}\|$
$\|\mathbf{U}\mathbf{U}^\top \mathbf{x}\|$

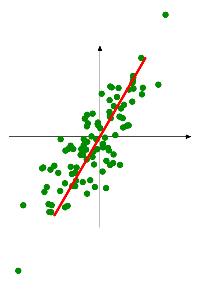
Take expectations; note rotation $\mathbf{U}$ doesn't affect length:

$$\hat{\mathbb{E}}[\|\mathbf{x}\|^2] = \hat{\mathbb{E}}[\|\mathbf{U}^\top \mathbf{x}\|^2] + \hat{\mathbb{E}}[\|\mathbf{x} - \mathbf{U}\mathbf{U}^\top \mathbf{x}\|^2]$$

Minimize reconstruction error $\leftrightarrow$ Maximize captured variance

Surprise—it turns out that the two perspectives on PCA are equivalent.

Finding one principal component



Input data:

$$\mathbf{X} = \begin{pmatrix} | & & | \\ \mathbf{x}_1 & \cdots & \mathbf{x}_n \\ | & & | \end{pmatrix}$$

Objective: maximize variance of projected data

$$= \max_{\|\mathbf{u}\|=1} \hat{\mathbb{E}}[(\mathbf{u}^\top \mathbf{x})^2]$$

$$= \max_{\|\mathbf{u}\|=1} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{u}^\top \mathbf{x}_i)^2$$

$$= \max_{\|\mathbf{u}\|=1} \frac{1}{n} \|\mathbf{u}^\top \mathbf{X}\|^2$$

$$= \max_{\|\mathbf{u}\|=1} \mathbf{u}^\top \left(\frac{1}{n}\mathbf{X}\mathbf{X}^\top\right) \mathbf{u}$$

$= \text{largest eigenvalue of } C \overset{\text{def}}{=} \frac{1}{n}\mathbf{X}\mathbf{X}^\top$

(C is covariance matrix of data)

Now let's start thinking about how to solve it. For this, it's most convenient to use the maximum variance perspective. First, consider reducing the number of dimensions down to $k = 1$. This short derivation shows how the eigenvalue problem arises.

Derivation for one principal component PCA

The first principal component can be expressed as an optimization problem (this is the variational formulation). We can remove the norm constraint by explicitly normalizing in the objective:

$$\max_{\|\mathbf{u}\|=1} \|\mathbf{u}^\top \mathbf{X}\|^2 = \max_{\mathbf{u}} \frac{\mathbf{u}^\top \mathbf{X}\mathbf{X}^\top \mathbf{u}}{\mathbf{u}^\top \mathbf{u}}$$

Let $(\lambda_1, \mathbf{u}_1), \dots (\lambda_1, \mathbf{u}_n)$ be the eigenvalues and eigenvectors of $\mathbf{X}\mathbf{X}^\top$. Each vector $\mathbf{u}$ has an eigendecomposition $\mathbf{u} = \sum_i a_i \mathbf{u}_i$, so we have the following equivalent optimization problem:

$$\max_{\mathbf{a}} \frac{(\sum_i a_i \mathbf{u}_i)^\top \mathbf{X}\mathbf{X}^\top (\sum_i a_i \mathbf{u}_i)}{(\sum_i a_i \mathbf{u}_i)^\top (\sum_i a_i \mathbf{u}_i)}$$

Using the fact that $\mathbf{u}_i^\top \mathbf{u}_j = 1$ if $i = j$ (and 0 otherwise) and $\mathbf{X}\mathbf{X}^\top \mathbf{u}_i = \lambda_i \mathbf{u}_i$, we simply to the following:

$$\max_{\mathbf{a}} \frac{\sum_i a_i^2 \lambda_i}{\sum_i a_i^2}.$$

If we think of the $\mathbf{a}_i$'s as specifying a distribution over the eigenvectors, the above quantity is clearly maximized when all the mass is placed on the largest eigenvector, which is $\mathbf{a} = (1, 0, 0, \dots)$, corresponding to $\mathbf{u} = \mathbf{u}_1$.

Another way to see the eigenvalue solution is in terms of Lagrange multipliers. We start out with the same constrained optimization problem. (Note that replacing $\|\mathbf{u}\| = 1$ with $\|\mathbf{u}\|^2 \le 1$ does not affect the solution. Why?)

$$\text{maximize } \|\mathbf{u}^\top \mathbf{X}\|^2 \text{ subject to } \|\mathbf{u}\|^2 \le 1.$$

We construct the Lagrangian:

$$L(\mathbf{u}, \lambda) = \|\mathbf{u}^\top \mathbf{X}\|^2 + \lambda(\|\mathbf{u}\|^2 - 1).$$

This is a convex optimization problem, so taking the gradient with respect to $\mathbf{u}$ and setting it to zero gives a sufficient condition for optimality:

$$\nabla L(\mathbf{u}, \lambda) = 2(\mathbf{X}\mathbf{X}^\top)\mathbf{u} - 2\lambda \mathbf{u} = 0.$$

Rewriting the above expression reveals that it is just an eigenvalue problem:

$$(\mathbf{X}\mathbf{X}^\top)\mathbf{u} = \lambda \mathbf{u}.$$

Equivalence to minimizing reconstruction error

We will show that maximizing the variance along the principal component is equivalent to minimizing the reconstruction error, i.e., the sum of the squares of the perpendicular distance from the component:

$$\begin{aligned}
\text{Reconstruction error} &= \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{u}\mathbf{u}^\top \mathbf{x}_i\|^2 \\
&= \sum_{i=1}^n \left(\|\mathbf{x}_i\|^2 - (\mathbf{u}^\top \mathbf{x}_i)^2\right) \\
&= \text{constant} - \sum_{i=1}^n (\mathbf{u}^\top \mathbf{x}_i)^2 \\
&= \text{constant} - \|\mathbf{u}^\top \mathbf{X}\|^2
\end{aligned}$$

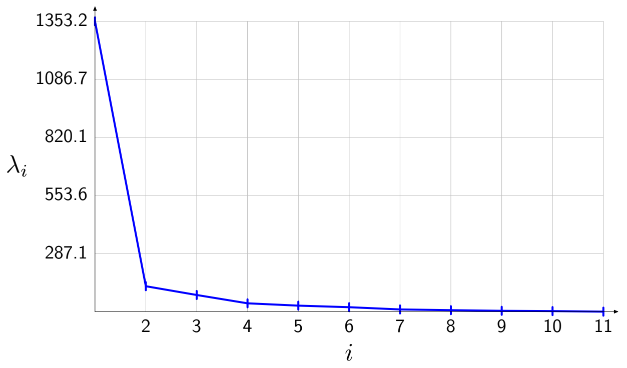
Note that $\mathbf{u} \perp (\mathbf{x}_i - \mathbf{u}\mathbf{u}^\top \mathbf{x}_i)$ because

$$\mathbf{u}^\top (\mathbf{x}_i - \mathbf{u}\mathbf{u}^\top \mathbf{x}_i) = 0,$$

so the second line follows from Pythagoras's theorem. These derivations show that minimizing reconstruction error is the same as maximizing variance.

How many principal components?

- Similar to question of “How many clusters?”
- Magnitude of eigenvalues indicate fraction of variance captured.
- Eigenvalues on a face image dataset:



- Eigenvalues typically drop off sharply, so don’t need that many.
- Of course variance isn’t everything...

The total variance is the sum of all the eigenvalues, which is just the trace of the covariance matrix (sum of diagonal entries). For typical data sets, the eigenvalues decay rapidly.

Computing PCA

Method 1: eigendecomposition

$\mathbf{U}$ are eigenvectors of covariance matrix $C = \frac{1}{n}\mathbf{X}\mathbf{X}^\top$

Computing C already takes $O(nd^2)$ time (very expensive)

Method 2: singular value decomposition (SVD)

Find $\mathbf{X} = \mathbf{U}_{d\times d}\Sigma_{d\times n}\mathbf{V}^\top_{n\times n}$

where $\mathbf{U}^\top\mathbf{U} = I_{d\times d}$, $\mathbf{V}^\top\mathbf{V} = I_{n\times n}$, Σ is diagonal

Computing top k singular vectors takes only $O(ndk)$

Relationship between eigendecomposition and SVD:

Left singular vectors are principal components ($C = \mathbf{U}\Sigma^2\mathbf{U}^\top$)

There are (at least) two ways to solve the eigenvalue problem. Just computing the covariance matrix C can be too expensive, so it’s usually better to go with the SVD (one line of Matlab).

Roadmap

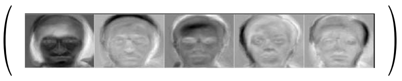
- Principal component analysis (PCA)
  - Basic principles
  - Case studies
  - Kernel PCA
  - Probabilistic PCA
- Canonical correlation analysis (CCA)
- Fisher discriminant analysis (FDA)
- Summary

Eigen-faces [Turk and Pentland, 1991]

- d = number of pixels
- Each $\mathbf{x}_i \in \mathbb{R}^d$ is a face image
- $\mathbf{x}_{ji}$ = intensity of the j-th pixel in image i

$$\mathbf{X}_{d\times n} \approx \mathbf{U}_{d\times k} \quad \mathbf{Z}_{k\times n}$$

$\approx$



$$\begin{pmatrix} | & & | \\ \mathbf{z}_1 & \cdots & \mathbf{z}_n \\ | & & | \end{pmatrix}$$

Idea: $\mathbf{z}_i$ more "meaningful" representation of i-th face than $\mathbf{x}_i$

Can use $\mathbf{z}_i$ for nearest-neighbor classification

Much faster: $O(dk + nk)$ time instead of $O(dn)$ when $n, d \gg k$

Why no time savings for linear classifier?

An application of PCA is image analysis. Here, the principal components (eigenvectors) are images that resemble faces. Note that the pixel representation is sensitive to rotation and translation (in image space). One can store $\mathbf{z}$ as a compressed version of the original image $\mathbf{x}$. The $\mathbf{z}$ can be used as features in classification.

Latent Semantic Analysis [Deerwater, 1990]

- d = number of words in the vocabulary
- Each $\mathbf{x}_i \in \mathbb{R}^d$ is a vector of word counts
- $\mathbf{x}_{ji}$ = frequency of word j in document i

$$\mathbf{X}_{d\times n} \approx \mathbf{U}_{d\times k} \quad \mathbf{Z}_{k\times n}$$

$$\left(\begin{array}{rccc} \text{stocks:} & 2 & \cdots\cdots\cdots & 0 \\ \text{chairman:} & 4 & \cdots\cdots\cdots & 1 \\ \text{the:} & 8 & \cdots\cdots\cdots & 7 \\ \cdots & \vdots & \cdots\cdots\cdots & \vdots \\ \text{wins:} & 0 & \cdots\cdots\cdots & 2 \\ \text{game:} & 1 & \cdots\cdots\cdots & 3 \end{array}\right) \approx \left(\begin{array}{ccc} 0.4 & \cdot\cdot & -0.001 \\ 0.8 & \cdot\cdot & 0.03 \\ 0.01 & \cdot\cdot & 0.04 \\ \vdots & \cdot\cdot & \vdots \\ 0.002 & \cdot\cdot & 2.3 \\ 0.003 & \cdot\cdot & 1.9 \end{array}\right) \left(\begin{array}{ccc} | & & | \\ \mathbf{z}_1 & \cdots & \mathbf{z}_n \\ | & & | \end{array}\right)$$

How to measure similarity between two documents?

$\mathbf{z}_1^\top \mathbf{z}_2$ is probably better than $\mathbf{x}_1^\top \mathbf{x}_2$

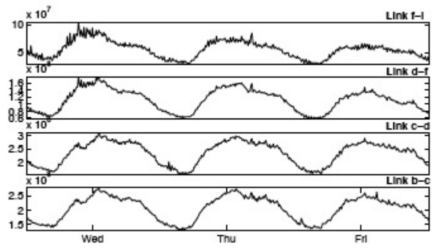
Applications: information retrieval

Note: no computational savings; original $\mathbf{x}$ is already sparse

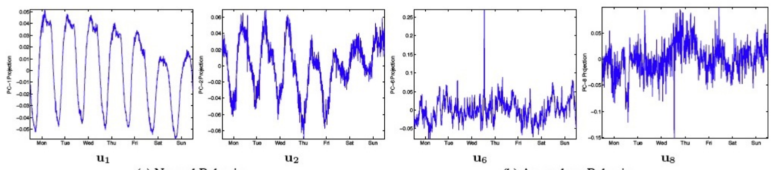
Latent Semantic Analysis (LSA), also known as Latent Semantic Indexing, is an application of PCA to categorical data. LSA is often used in information retrieval. Eigen-documents tries to capture "semantics": an eigen-document contains related words. But how do we interpret negative frequencies? Other methods such as probabilistic LSA, Latent Dirichlet Allocation, or non-negative matrix factorization may lead to more interpretable results.

Network anomaly detection [Lakhina, '05]

$\mathbf{x}_{ji}$ = amount of traffic on link j in the network during each time interval i



Model assumption: total traffic is sum of flows along a few "paths"
Apply PCA: each principal component intuitively represents a "path"
Anomaly when traffic deviates from first few principal components



(a) Normal Behavior (b) Anomalous Behavior

In this application, PCA is used more directly to model the data. Each data point is a snapshot of the network at some point in time. Of course principal components won't be actual paths, but they will represent network links which tend to be correlated. If at some point in time at test time, the reconstruction error of a test point is high, raise a red flag.

Unsupervised POS tagging [Schütze, '95]

Part-of-speech (POS) tagging task:

Input:	I	like	reducing	the	dimensionality	of	data	.
Output:	NOUN	VERB	VERB(-ING)	DET	NOUN	PREP	NOUN	.

Each $\mathbf{x}_i$ is (the context distribution of) a word.

$\mathbf{x}_{ji}$ is number of times word i appeared in context j

Key idea: words appearing in similar contexts
tend to have the same POS tags;
so cluster using the contexts of each word type

Problem: contexts are too sparse

Solution: run PCA first,
then cluster using new representation

Here, PCA is used as a preprocessing step to fight the curse of dimensionality typical in natural language (not enough data points).

Multi-task learning [Ando & Zhang, '05]

- Have n related tasks (classify documents for various users)
- Each task has a linear classifier with weights $\mathbf{x}_i$
- Want to share structure between classifiers

One step of their procedure:
given n linear classifiers $\mathbf{x}_1, \dots, \mathbf{x}_n$,
run PCA to identify shared structure:

$$\mathbf{X} = \begin{pmatrix} | & & | \\ \mathbf{x}_1 & \dots & \mathbf{x}_n \\ | & & | \end{pmatrix} \approx \mathbf{UZ}$$

Each principal component is a eigen-classifier

Other step of their procedure:
Retrain classifiers, regularizing towards subspace $\mathbf{U}$

This is a neat application of PCA which is more abstract than the previous ones. It can be applied in many types of general machine learning scenarios.

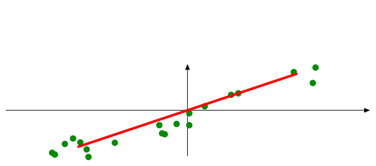
PCA summary

- Intuition: capture variance of data or minimize reconstruction error
- Algorithm: find eigendecomposition of covariance matrix or SVD
- Impact: reduce storage (from $O(nd)$ to $O(nk)$), reduce time complexity
- Advantages: simple, fast
- Applications: eigen-faces, eigen-documents, network anomaly detection, etc.

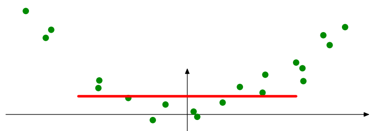
Roadmap

- Principal component analysis (PCA)
  - Basic principles
  - Case studies
  - Kernel PCA
  - Probabilistic PCA
- Canonical correlation analysis (CCA)
- Fisher discriminant analysis (FDA)
- Summary

Limitations of linearity



PCA is effective



PCA is ineffective

Problem is that PCA subspace is linear:

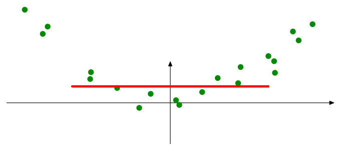
$$S = \{\mathbf{x} = \mathbf{U}\mathbf{z} : \mathbf{z} \in \mathbb{R}^k\}$$

In this example:

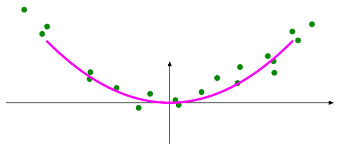
$$S = \{(x_1, x_2) : x_2 = \tfrac{u_2}{u_1} x_1\}$$

Remember that PCA can only find linear subspaces. But this lecture is only about linear dimensionality reduction...or is it? S is the subspace, the set of points formed by linear combinations of the principal components. The second way to write it exposes subspace constraint directly.

Going beyond linearity: quick solution



Broken solution



Desired solution

We want desired solution: $S = \{(x_1, x_2) : x_2 = \frac{u_2}{u_1} x_1^2\}$

We can get this: $S = \{\phi(\mathbf{x}) = \mathbf{U}\mathbf{z}\}$ with $\phi(\mathbf{x}) = (x_1^2, x_2)^\top$

$$\boxed{\begin{array}{c}\text{Linear dimensionality reduction in } \phi(\mathbf{x}) \text{ space} \\ \Updownarrow \\ \text{Nonlinear dimensionality reduction in } \mathbf{x} \text{ space}\end{array}}$$

In general, can set $\phi(\mathbf{x}) = (x_1, x_1^2, x_1x_2, \sin(x_1), \dots)^\top$

Problems: (1) ad-hoc and tedious
(2) $\phi(\mathbf{x})$ large, computationally expensive

Remember that, as we saw in linear regression, linear means linear in the parameters, not linear in the features, which can be anything you want, in particular, $\phi(\mathbf{x})$. But we want to include all quadratic terms, that's $O(d^2)$ of them, which is very expensive. Intuitively, we should never need to work with more dimensions than the number of data points...

Towards kernels

Representer theorem:

PCA solution is linear combination of $\mathbf{x}_i$s

Why?

Recall PCA eigenvalue problem: $\mathbf{X}\mathbf{X}^\top \mathbf{u} = \lambda \mathbf{u}$

Notice that $\mathbf{u} = \mathbf{X}\boldsymbol{\alpha} = \sum_{i=1}^n \alpha_i \mathbf{x}_i$ for some weights $\boldsymbol{\alpha}$

Analogy with SVMs: weight vector $\mathbf{w} = \mathbf{X}\alpha$

Key fact:

PCA only needs inner products $K = \mathbf{X}^\top \mathbf{X}$

Why?

Use representer theorem on PCA objective:

$$\max_{\|\mathbf{u}\|=1} \mathbf{u}^\top \mathbf{X}\mathbf{X}^\top \mathbf{u} = \max_{\boldsymbol{\alpha}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\alpha}=1} \boldsymbol{\alpha}^\top (\mathbf{X}^\top \mathbf{X})(\mathbf{X}^\top \mathbf{X})\boldsymbol{\alpha}$$

Let's try to see if we can approach PCA from a data-point-centric perspective. There are two logical steps here. The main take-away is that PCA only needs inner products, and inner products can be computed without instantiating the original vectors.

Kernel PCA

Kernel function: $k(\mathbf{x}_1, \mathbf{x}_2)$ such that

K, the kernel matrix formed by $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$,

is positive semi-definite

Examples:

Linear kernel: $k(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^\top \mathbf{x}_2$

Polynomial kernel: $k(\mathbf{x}_1, \mathbf{x}_2) = (1 + \mathbf{x}_1^\top \mathbf{x}_2)^2$

Gaussian (RBF) kernel: $k(\mathbf{x}_1, \mathbf{x}_2) = e^{-\|\mathbf{x}_1 - \mathbf{x}_2\|^2}$

Treat data points $\mathbf{x}$ as black boxes, only access via k

k intuitively measures "similarity" between two inputs

Mercer's theorem (using kernels is sensible)

Exists high-dimensional feature space ϕ such that

$k(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_2)$ (like quick solution earlier!)

Kernels allow you to directly compute these inner products. The punchline is Mercer's theorem, which says that by using kernels, we are implicitly working in the high-dimensional space $\phi(\mathbf{x})$ (which we tried to construct manually before) which might be infinite dimensional. Some guidance: if you know exactly what kind of non-linearity you need, you can just add features to ϕ directly (feature engineering). Kernels let you do this more in a more automatic way, but the straightforward approach takes $O(n^3)$ time.

Solving kernel PCA

Direct method:

Kernel PCA objective:

$$\max_{\boldsymbol{\alpha}^\top K \boldsymbol{\alpha}=1} \boldsymbol{\alpha}^\top K^2 \boldsymbol{\alpha}$$

$\Rightarrow$ kernel PCA eigenvalue problem: $\mathbf{X}^\top \mathbf{X}\boldsymbol{\alpha} = \lambda' \boldsymbol{\alpha}$

Modular method (if you don't want to think about kernels):

Find vectors $\mathbf{x}'_1, \dots, \mathbf{x}'_n$ such that

$$\mathbf{x}'^\top_i \mathbf{x}'_j = K_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

Key: use any vectors that preserve inner products

One possibility is Cholesky decomposition $K = \mathbf{X}^\top \mathbf{X}$

Most textbooks will talk about the direct method. The modular method capitalizes on the fact that only inner products matter. So we can rotate our points any way we'd like as long as we preserve inner products (treat the point set as a rigid body). The method decouples into two steps: (1) find a n dimensional space to work in, and (2) do regular PCA in that space. Note that you only have to talk about kernels in the first step. This way, we never have to write new code to kernelize a linear method (PCA, CCA, FDA, SVMs, logistic regression, etc.)

Roadmap

- Principal component analysis (PCA)
  - Basic principles
  - Case studies
  - Kernel PCA
  - Probabilistic PCA
- Canonical correlation analysis (CCA)
- Fisher discriminant analysis (FDA)
- Summary

Probabilistic modeling

So far, deal with objective functions:

$$\min_{\mathbf{U}} f(\mathbf{X}, \mathbf{U})$$

Probabilistic modeling:

$$\max_{\mathbf{U}} p(\mathbf{X} \mid \mathbf{U})$$

Invent a generative story of how data $\mathbf{X}$ arose
Play detective: infer parameters $\mathbf{U}$ that produced $\mathbf{X}$

Advantages:

- Model reports estimates of uncertainty
- Natural way to handle missing data
- Natural way to introduce prior knowledge
- Natural way to incorporate in a larger model

Example from last lecture: k-means $\Rightarrow$ GMMs

Probabilistic modeling is a mode of thinking, a language, which sits at a higher level of abstraction than objective functions, which is higher up than reasoning directly about algorithms.

Probabilistic PCA

Generative story [Tipping and Bishop, 1999]:

```
For each data point $i = 1, \dots, n$:
  Draw the latent vector: $\mathbf{z}_i \sim \mathcal{N}(0, I_{k \times k})$
  Create the data point: $\mathbf{x}_i \sim \mathcal{N}(\mathbf{U}\mathbf{z}_i, \sigma^2 I_{d \times d})$
```

PCA finds the $\mathbf{U}$ that maximizes the likelihood of the data

Advantages:

- Handles missing data (important for collaborative filtering)
- Extension to factor analysis: allow non-isotropic noise (replace $\sigma^2 I_{d \times d}$ with arbitrary diagonal matrix)

In many cases, the maximum likelihood estimate of a model is equivalent to a natural objective function.

Probabilistic latent semantic analysis (pLSA)

Motivation: in text analysis, $\mathbf{X}$ contains word counts; PCA (LSA) is bad model as it allows negative counts; pLSA fixes this

Generative story for pLSA [Hofmann, 1999]:

```
For each document $i = 1, \dots, n$:
  Repeat $M$ times (number of word tokens in document):
    Draw a latent topic: $\mathbf{z} \sim p(\mathbf{z} \mid i)$
    Choose the word token: $\mathbf{x} \sim p(\mathbf{x} \mid \mathbf{z})$
  Set $\mathbf{x}_{ji}$ to be the number of times word $j$ was chosen
```

Learning using Hard EM (analog of k-means):
  E-step: fix parameters, choose best topics
  M-step: fix topics, optimize parameters

More sophisticated methods: EM, Latent Dirichlet Allocation

Comparison to a mixture model for clustering:
  Mixture model: assume a single topic for entire document
  pLSA: allow multiple topics per document

By thinking probabilistically, it's natural to adapt probabilistic PCA to better fit the type of data.

Roadmap

- Principal component analysis (PCA)
  - Basic principles
  - Case studies
  - Kernel PCA
  - Probabilistic PCA
- Canonical correlation analysis (CCA)
- Fisher discriminant analysis (FDA)
- Summary

Motivation for CCA [Hotelling, 1936]

Often, each data point consists of two views:

- Image retrieval: for each image, have the following:
  - $\mathbf{x}$: Pixels (or other visual features)
  - $\mathbf{y}$: Text around the image
- Time series:
  - $\mathbf{x}$: Signal at time t
  - $\mathbf{y}$: Signal at time $t+1$
- Two-view learning: divide features into two sets
  - $\mathbf{x}$: Features of a word/object, etc.
  - $\mathbf{y}$: Features of the context in which it appears

Goal: reduce the dimensionality of the two views jointly

Each data point is a vector, which can be broken up into two parts (possibly of different dimensionality), which we call $\mathbf{x}$ and $\mathbf{y}$. Two-view learning is another abstract technique in machine learning where we assume each input point can be split up into two types of features, each of which can do a reasonable job of predicting the output, but are in some ways complementary. One can use CCA to reduce the dimensionality of the two views (leading to better statistical generalization) while staying relevant to the prediction problem.

An example

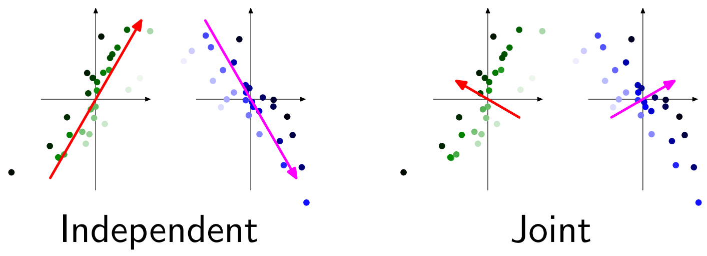
Setup:

Input data: $(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)$ (matrices $\mathbf{X}, \mathbf{Y}$)

Goal: find pair of projections $(\mathbf{u}, \mathbf{v})$

In figure, $\mathbf{x}$ and $\mathbf{y}$ are paired by brightness

Dimensionality reduction solutions:



Here, we'd like an algorithm that chooses directions jointly so that dark points have large projected coordinates and light points have small projected coordinates, both for the $\mathbf{x}$ view and the $\mathbf{y}$ view, causing them to be correlated.

From PCA to CCA

PCA on views separately: no covariance term

$$\max_{\mathbf{u},\mathbf{v}} \frac{\mathbf{u}^\top \mathbf{X}\mathbf{X}^\top \mathbf{u}}{\mathbf{u}^\top \mathbf{u}} + \frac{\mathbf{v}^\top \mathbf{Y}\mathbf{Y}^\top \mathbf{v}}{\mathbf{v}^\top \mathbf{v}}$$

PCA on concatenation $(\mathbf{X}^\top, \mathbf{Y}^\top)^\top$: includes covariance term

$$\max_{\mathbf{u},\mathbf{v}} \frac{\mathbf{u}^\top \mathbf{X}\mathbf{X}^\top \mathbf{u} + 2\mathbf{u}^\top \mathbf{X}\mathbf{Y}^\top \mathbf{v} + \mathbf{v}^\top \mathbf{Y}\mathbf{Y}^\top \mathbf{v}}{\mathbf{u}^\top \mathbf{u} + \mathbf{v}^\top \mathbf{v}}$$

Maximum covariance: drop variance terms

$$\max_{\mathbf{u},\mathbf{v}} \frac{\mathbf{u}^\top \mathbf{X}\mathbf{Y}^\top \mathbf{v}}{\sqrt{\mathbf{u}^\top \mathbf{u}}\sqrt{\mathbf{v}^\top \mathbf{v}}}$$

Maximum correlation (CCA): divide out variance terms

$$\max_{\mathbf{u},\mathbf{v}} \frac{\mathbf{u}^\top \mathbf{X}\mathbf{Y}^\top \mathbf{v}}{\sqrt{\mathbf{u}^\top \mathbf{X}\mathbf{X}^\top \mathbf{u}}\sqrt{\mathbf{v}^\top \mathbf{Y}\mathbf{Y}^\top \mathbf{v}}}$$

Let's morph PCA on the two views independently into CCA, which focuses exclusively on the correlation.

Canonical correlation analysis (CCA)

Definitions:

Variance: $\widehat{\text{var}}(\mathbf{u}^\top \mathbf{x}) = \mathbf{u}^\top \mathbf{X}\mathbf{X}^\top \mathbf{u}$

Covariance: $\widehat{\text{cov}}(\mathbf{u}^\top \mathbf{x}, \mathbf{v}^\top \mathbf{y}) = \mathbf{u}^\top \mathbf{X}\mathbf{Y}^\top \mathbf{v}$

Correlation: $\frac{\widehat{\text{cov}}(\mathbf{u}^\top \mathbf{x}, \mathbf{v}^\top \mathbf{y})}{\sqrt{\widehat{\text{var}}(\mathbf{u}^\top \mathbf{x})}\sqrt{\widehat{\text{var}}(\mathbf{v}^\top \mathbf{y})}}$

Objective: maximize correlation between projected views

$$\max_{\mathbf{u},\mathbf{v}} \widehat{\text{corr}}(\mathbf{u}^\top \mathbf{x}, \mathbf{v}^\top \mathbf{y})$$

Properties:

- Focus on how variables are related, not how much they vary
- Invariant to any rotation and scaling of data

Solved via a generalized eigenvalue problem ($A\mathbf{w} = \lambda B\mathbf{w}$)

CCA maximizes the correlation between two views. Recall that correlation is always between -1 and 1.

CCA objective function

Objective: maximize correlation between projected views

$$= \max_{\mathbf{u},\mathbf{v}} \widehat{\mathsf{corr}}(\mathbf{u}^\top \mathbf{x}, \mathbf{v}^\top \mathbf{y}) = \max_{\mathbf{u},\mathbf{v}} \frac{\widehat{\mathsf{cov}}(\mathbf{u}^\top \mathbf{x}, \mathbf{v}^\top \mathbf{y})}{\sqrt{\widehat{\mathsf{var}}(\mathbf{u}^\top \mathbf{x})}\sqrt{\widehat{\mathsf{var}}(\mathbf{v}^\top \mathbf{y})}}$$

$$= \max_{\widehat{\mathsf{var}}(\mathbf{u}^\top \mathbf{x})=\widehat{\mathsf{var}}(\mathbf{v}^\top \mathbf{y})=1} \widehat{\mathsf{cov}}(\mathbf{u}^\top \mathbf{x}, \mathbf{v}^\top \mathbf{y})$$

$$= \max_{\|\mathbf{u}^\top \mathbf{X}\|=\|\mathbf{v}^\top \mathbf{Y}\|=1} \sum_{i=1}^{n} (\mathbf{u}^\top \mathbf{x}_i)(\mathbf{v}^\top \mathbf{y}_i)$$

$$= \max_{\|\mathbf{u}^\top \mathbf{X}\|=\|\mathbf{v}^\top \mathbf{Y}\|=1} \mathbf{u}^\top \mathbf{X}\mathbf{Y}^\top \mathbf{v}$$

= largest generalized eigenvalue λ given by

$$\begin{pmatrix} 0 & \mathbf{X}\mathbf{Y}^\top \\ \mathbf{Y}\mathbf{X}^\top & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{X}\mathbf{X}^\top & 0 \\ 0 & \mathbf{Y}\mathbf{Y}^\top \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix},$$

which reduces to an ordinary eigenvalue problem.

Derivation for CCA (one component)

The first principal component is given by solving the following objective function:

$$\text{maximize } \mathbf{u}^\top \mathbf{X}\mathbf{Y}^\top \mathbf{v} \text{ subject to } \mathbf{u}^\top \mathbf{X}\mathbf{X}^\top \mathbf{u} \le 1 \text{ and } \mathbf{v}^\top \mathbf{Y}\mathbf{Y}^\top \mathbf{v} \le 1.$$

This is a constrained optimization problem, so we construct the Lagrangian:

$$L(\mathbf{u}, \mathbf{v}, \lambda_u, \lambda_v) = \mathbf{u}^\top \mathbf{X}\mathbf{Y}^\top \mathbf{v} + \lambda_u(\mathbf{u}^\top \mathbf{X}\mathbf{X}^\top \mathbf{u} - 1) + \lambda_v(\mathbf{v}^\top \mathbf{Y}\mathbf{Y}^\top \mathbf{v} - 1).$$

This is a convex optimization problem, so taking the gradient with respect to $\mathbf{u}$ and $\mathbf{v}$ and setting them to zero gives a sufficient condition for optimality:

$$\nabla L(\mathbf{u}, \mathbf{v}, \lambda_u, \lambda_v) = \begin{pmatrix} (\mathbf{Y}\mathbf{X}^\top)\mathbf{u} - 2\lambda_v(\mathbf{Y}\mathbf{Y}^\top)\mathbf{v} \\ (\mathbf{X}\mathbf{Y}^\top)\mathbf{v} - 2\lambda_u(\mathbf{X}\mathbf{X}^\top)\mathbf{u} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Left-multiplying the top row by $\mathbf{v}^\top$ and the bottom row by $\mathbf{u}^\top$ and subtracting results in

$$\lambda_u \mathbf{u}^\top (\mathbf{X}\mathbf{X}^\top)\mathbf{u} = \lambda_v \mathbf{v}^\top (\mathbf{Y}\mathbf{Y}^\top)\mathbf{v}.$$

Since $\mathbf{u}^\top(\mathbf{X}\mathbf{X}^\top)\mathbf{u} = \mathbf{v}^\top(\mathbf{Y}\mathbf{Y}^\top)\mathbf{v} = 1$ at the optimum, $\lambda_u = \lambda_v \overset{\text{def}}{=} \lambda$. Now we can write the optimality conditions as a generalized eigenvalue problem:

$$\begin{pmatrix} 0 & \mathbf{X}\mathbf{Y}^\top \\ \mathbf{Y}\mathbf{X}^\top & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{X}\mathbf{X}^\top & 0 \\ 0 & \mathbf{Y}\mathbf{Y}^\top \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}.$$

Regularization is important

Extreme examples of degeneracy:

- If $\mathbf{x} = A\mathbf{y}$, then any $(\mathbf{u}, \mathbf{v})$ with $\mathbf{u} = A\mathbf{v}$ is optimal (correlation 1)
- If $\mathbf{x}$ and $\mathbf{y}$ are independent, then any $(\mathbf{u}, \mathbf{v})$ is optimal (correlation 0)

Problem: if $\mathbf{X}$ or $\mathbf{Y}$ has rank n, then any $(\mathbf{u}, \mathbf{v})$ is optimal (correlation 1) with $\mathbf{u} = \mathbf{X}^{\dagger\top}\mathbf{Y}\mathbf{v} \Rightarrow$ CCA is meaningless!

Solution: regularization (interpolate between maximum covariance and maximum correlation)

$$\max_{\mathbf{u},\mathbf{v}} \frac{\mathbf{u}^\top \mathbf{X}\mathbf{Y}^\top \mathbf{v}}{\sqrt{\mathbf{u}^\top(\mathbf{X}\mathbf{X}^\top + \lambda I)\mathbf{u}}\sqrt{\mathbf{v}^\top(\mathbf{Y}\mathbf{Y}^\top + \lambda I)\mathbf{v}}}$$

Regularization is very important in machine learning to limit the capacity of a model (e.g., penalize by the norm of the weight vector). For PCA, we are in some sense regularizing by virtue of choosing a small k. The key point about CCA is that this is not enough, even when $k = 1$! This is a problem when either view has rank n (typically, $d \gg n$). In this case, we can basically set $\mathbf{u}$ to anything. Note that CCA is very sensitive to noise—even the smallest amount of variation gets normalized to 1. Regularizing towards maximum covariance forces us to break ties by consider which projections are explaining the variation in the data better.

Kernel CCA

Two kernels: k_x and k_y

Direct method:

(some math)

Modular method:

1. Transform $\mathbf{x}_i$ into $\mathbf{x}'_i \in \mathbb{R}^n$ satisfying $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}'^\top_i \mathbf{x}'_j$ (do same for $\mathbf{y}$)
2. Perform regular CCA

Regularization is especially important for kernel CCA!

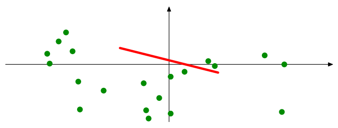
If you use the Gaussian kernel, the kernel matrix will always have rank n, so you better regularize.

Roadmap

- Principal component analysis (PCA)
  - Basic principles
  - Case studies
  - Kernel PCA
  - Probabilistic PCA
- Canonical correlation analysis (CCA)
- Fisher discriminant analysis (FDA)
- Summary

Motivation for FDA [Fisher, 1936]

What is the best linear projection?

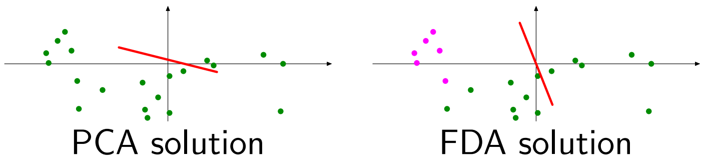


PCA solution

Interclass variance is the sum of squared distances of points in different classes; intraclass variance, of pairs of points in the same class. When one starts using labels, dimensionality reduction starts feeling like classification, which is an extreme form of dimensionality reduction tailored for prediction of a single variable.

Motivation for FDA [Fisher, 1936]

What is the best linear projection with these labels?



Goal: reduce the dimensionality given labels

Idea: want projection to maximize overall interclass variance relative to intraclass variance

Linear classifiers (logistic regression, SVMs) have similar feel:

Find one-dimensional subspace $\mathbf{w}$,

e.g., to maximize margin between different classes

FDA handles multiple classes, allows multiple dimensions

Interclass variance is the sum of squared distances of points in different classes; intraclass variance, of pairs of points in the same class. When one starts using labels, dimensionality reduction starts feeling like classification, which is an extreme form of dimensionality reduction tailored for prediction of a single variable.

FDA objective function

Setup: $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{1, \dots, m\}$, for $i = 1, \dots, n$

Objective: maximize $\frac{\text{interclass variance}}{\text{intraclass variance}} = \frac{\text{total variance}}{\text{intraclass variance}} - 1$

Total variance: $\frac{1}{n}\sum_i (\mathbf{u}^\top(\mathbf{x}_i - \mu))^2$

Mean of all points: $\mu = \frac{1}{n}\sum_i \mathbf{x}_i$

Intraclass variance: $\frac{1}{n}\sum_i (\mathbf{u}^\top(\mathbf{x}_i - \mu_{\mathbf{y}_i}))^2$

Mean of points in class y: $\mu_y = \frac{1}{|\{i : y_i = y\}|}\sum_{i : y_i = y} \mathbf{x}_i$

Reduces to a generalized eigenvalue problem.

Kernel FDA: use modular method

The total variance is the sum of interclass variance and intraclass variance.

FDA derivation

Global mean: $\mu = \sum_i \mathbf{x}_i$ $\quad \mathbf{X}_g = (\mathbf{x}_1 - \mu, \dots, \mathbf{x}_n - \mu)$

Class mean: $\mu_y = \sum_{i:y_i=y} \mathbf{x}_i$ $\quad \mathbf{X}_c = (\mathbf{x}_1 - \mu_{y_1}, \dots, \mathbf{x}_n - \mu_{y_n})$

Objective: maximize $\frac{\text{total variance}}{\text{intraclass variance}} = \frac{\text{interclass variance}}{\text{intraclass variance}} + 1$

$$= \max_{\mathbf{u}} \frac{\sum_{i=1}^n (\mathbf{u}^\top(\mathbf{x}_i - \mu))^2}{\sum_{i=1}^n (\mathbf{u}^\top(\mathbf{x}_i - \mu_{y_i}))^2}$$

$$= \max_{\|\mathbf{u}^\top \mathbf{X}_c\|=1} \sum_{i=1}^n (\mathbf{u}^\top(\mathbf{x}_i - \mu))^2$$

$$= \max_{\|\mathbf{u}^\top \mathbf{X}_c\|=1} \mathbf{u}^\top \mathbf{X}_g \mathbf{X}_g^\top \mathbf{u}$$

= largest generalized eigenvalue λ given by

$$(\mathbf{X}_g \mathbf{X}_g^\top)\mathbf{u} = \lambda(\mathbf{X}_c \mathbf{X}_c^\top)\mathbf{u}.$$

Other linear methods

Random projections:

Randomly project data onto $k = O(\log n)$ dimensions

All pairwise distances preserved with high probability

$$\|\mathbf{U}^\top \mathbf{x}_i - \mathbf{U}^\top \mathbf{x}_j\|^2 \approxeq \|\mathbf{x}_i - \mathbf{x}_j\|^2 \text{ for all } i, j$$

Trivial to implement

Kernel dimensionality reduction:

One type of sufficient dimensionality reduction

Find subspace that contains all information about labels

$$\mathbf{y} \perp\!\!\!\perp \mathbf{x} \mid \mathbf{U}^\top \mathbf{x}$$

Capturing information is stronger than capturing variance

Hard nonconvex optimization problem

Random projections is dead simple, has great theory, but in practice can be outperformed by smarter methods. Kernel dimensionality reduction is a much heavier duty, and it focuses on information/dependence rather than variance (recall two independent variables are uncorrelated but not vice-versa).

Summary

Framework: $\mathbf{z} = \mathbf{U}^\top \mathbf{x}$, $\mathbf{x} \approxeq \mathbf{U}\mathbf{z}$

Criteria for choosing $\mathbf{U}$:

- PCA: maximize projected variance
- CCA: maximize projected correlation
- FDA: maximize projected $\frac{\text{interclass variance}}{\text{intraclass variance}}$

Algorithm: generalized eigenvalue problem

Extensions:

non-linear using kernels (using same linear framework)

probabilistic, sparse, robust (hard optimization)

We focused on three very classical methods, all dating back at least 80 years. They are quite powerful and exploit the beauty of linear algebra. There are a number of extensions to meet various desiderata, but most of these extensions involve more difficult optimization problems and partially lose their elegance, though they can be effective in practice.