Announcements

- Only 131 people have signed up for a project team
  - if you have not signed up, or are on a team of 1, please try contacting other folks in the same situation
  - if this fails, please email me
- I will hold office hours Wednesday morning 9:00-10:30
- Proposal Deadline extended to March 16!

Dimensionality reduction

We observe data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

The goal of ***dimensionality reduction*** is to transform these inputs to new variables

$$\mathbf{x}_i \to \boldsymbol{\theta}_i \in \mathbb{R}^k$$

where $k \ll d$ in such a way that ***minimizes information loss***

Dimensionality reductions serves two main purposes:

- Helps (many) algorithms to be more computationally efficient
- Helps prevent overfitting (a form of regularization), especially when $n \leq d$

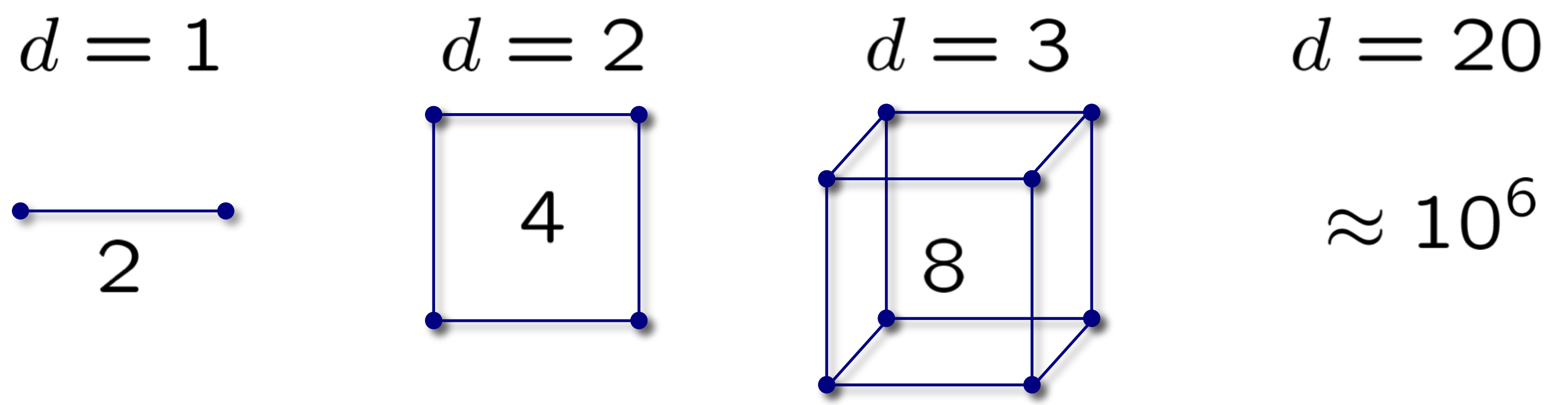
Curse of dimensionality

As the dimensionality of our feature space grows, the volume of the space increases…

A lot…

In learning, this often translates to requiring exponentially more data in order for the results to be reliable

Example: With binary features, how much data do we need to have at least one example of every possible combination of features?



Dimensionality reduction

Broadly speaking, methods for dimensionality reduction can be categorized according to:

1. How is "information loss" quantified?
2. Supervised or unsupervised?
   i.e., if labels $y_1, \ldots, y_n$ are available, how are they used?
3. Is the map $\mathbf{x} \to \boldsymbol{\theta}$ linear or nonlinear?
4. Feature ***selection*** versus feature ***extraction***?

$$\boldsymbol{\theta} = \begin{bmatrix} x(1) \\ x(7) \\ x(16) \\ \vdots \end{bmatrix} \quad \text{vs} \quad \boldsymbol{\theta} = \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \\ \phi_3(\mathbf{x}) \\ \vdots \end{bmatrix}$$

Feature selection

Feature ***selection*** is the problem of selecting a subset of the variables $x(1), \ldots, x(d)$ that are most relevant for a machine learning task (e.g., classification or regression)

Sometimes called ***subset selection***

There are three main reasons why we might want to perform feature selection:

- computational efficiency
- regularization
- retains interpretability

Feature selection (and feature extraction) improves performance by ***eliminating irrelevant features***

Filter methods

Filter methods attempt to ***rank*** features in order of importance and then take the top k features

In supervised learning, "importance" is usually related to the ability of a feature to ***predict*** the label or response variable

Advantage

- simple, fast

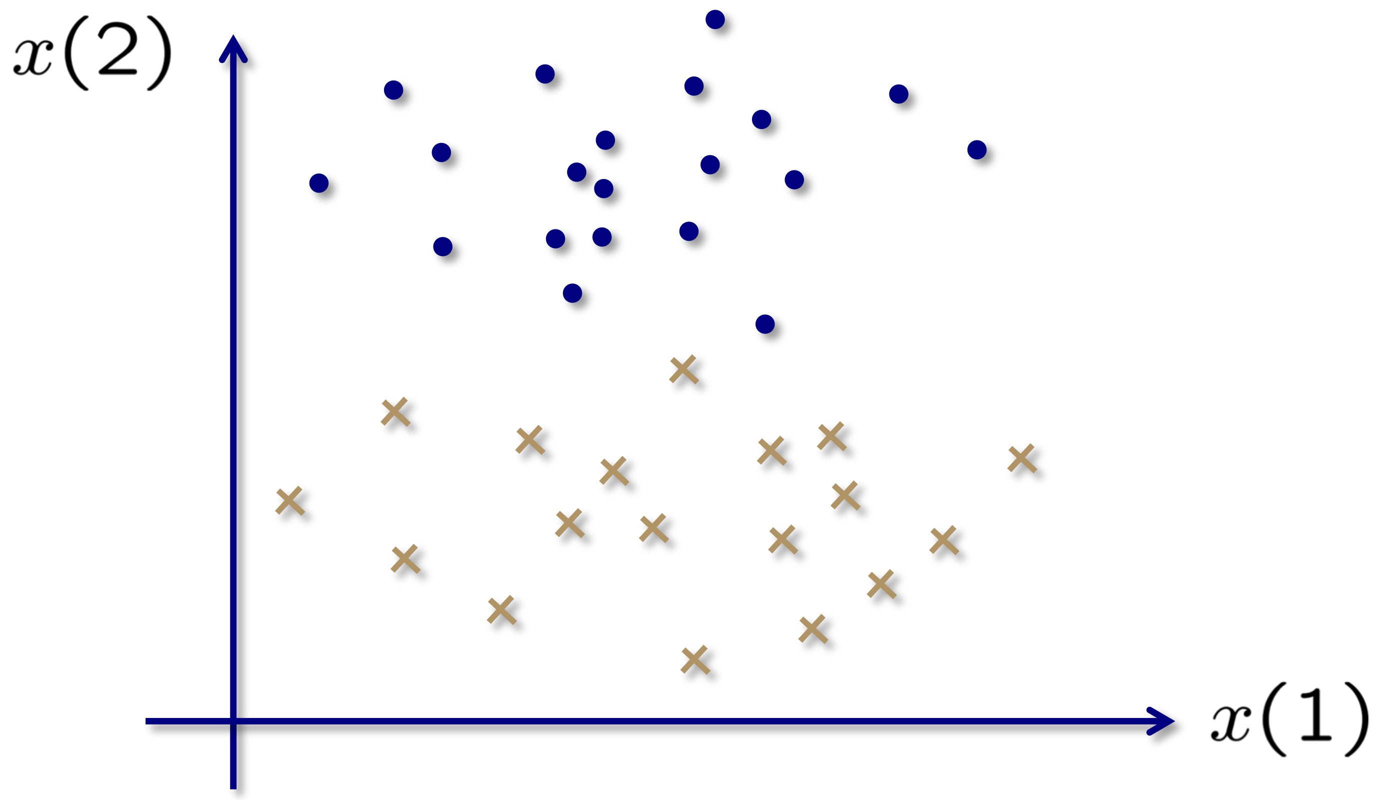
Disadvantage

- the k best features are usually not the best k features

The approach to ranking the features will depend on the application

Filtering in classification

Consider training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$



How should we rank the features?

Ranking criteria

Misclassification rate

$$r(j) = \frac{1}{n} \sum_{i=1}^{n} 1_{\{y_i \neq \theta(x_i(j))\}}$$

where θ is a classifier that compares the feature $x(j)$ to a threshold

Two sample t-test statistic

$$r(j) = \frac{\left| \overline{x_+(j)} - \overline{x_-(j)} \right|}{s/\sqrt{n}}$$

where $\overline{x_+(j)}$ and $\overline{x_-(j)}$ are the within-class means for feature $x(j)$ and s is the pooled sample standard deviation

Ranking criteria

Margin

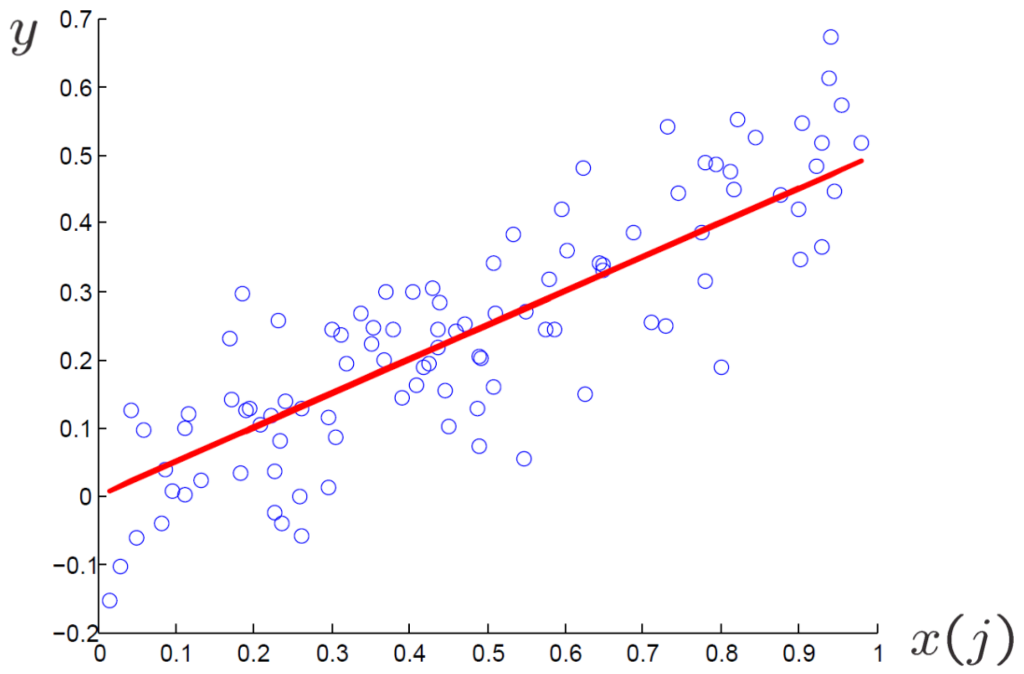
If the data is separable, then we can compute

$$r(j) = \min_{\substack{k: y_k = +1 \\ \ell: y_\ell = -1}} |x_k(j) - x_\ell(j)|$$

This can be made robust to the non-separable case by replacing the hard minimum with an ***order statistic*** that allows you to ignore some fixed number of outliers

Filtering in linear regression

In linear regression, we have training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where $y_i \in \mathbb{R}$, and we expect y to change linearly in response to changes in any $x(j)$



How should we rank the features?

Correlation coefficient

Pick the features which are ***most correlated*** with y

Set $r(j) = |\rho(j)|$ where

$$\rho(j) = \frac{\mathrm{cov}(x(j), y)}{\sqrt{\mathrm{var}(x(j)) \cdot \mathrm{var}(y)}}$$

$$= \frac{\sum_{i=1}^{n}(x_i(j) - \overline{x(j)})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i(j) - \overline{x(j)})^2 \cdot \sum_{i=1}^{n}(y_i - \overline{y})^2}}$$

Mutual information

The ***mutual information*** between X and Y is

$$I(X;Y) := \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

This is the Kullback-Leibler (KL) divergence between the joint distribution $p(x,y)$ and the product of the marginal distributions $p(x)p(y)$

Note that $I(X;Y) = 0$ if X are Y independent

You can intuitively think of $I(X;Y)$ as a measure of “how much knowing X tells us about Y”

Maximizing mutual information

If $X(S)$ denotes a subset of features corresponding to $S \subset \{1, \ldots, d\}$, then ideally we would like to maximize

$$I(X(S); Y)$$

over all possible S of a desired size

Unfortunately, this is typically intractable

Instead we could rank the features according to

$$r(j) = I(X(j); Y)$$

where the mutual information is estimated by first computing histograms or some other estimate of $p(x, y)$ and $p(x)p(y)$

Incremental maximization

This is a legitimate strategy, but (just like the other methods we have discussed) it can lead to selecting ***highly redundant*** features

With mutual information, there is a natural way to deal with this redundancy by selecting features ***incrementally***

For example, say that we have already selected features $X(j_1), \ldots, X(j_{k-1})$ and wish to select one more

Choose $X(j_k)$ to maximize

$$I(X(j_k);Y) - \beta \sum_{i=1}^{k-1} I(X(j_k);X(j_i))$$

Alternatives to filtering

A big drawback to the filtering approach is that it usually doesn't capture interactions between features

Can result in selecting ***redundant*** features

Wrapper methods are an alternative with three ingredients:

1. a machine learning algorithm
2. a way to assess the performance of a subset of features
3. a strategy for searching through subsets of features

Advantage

- captures feature interactions where filter methods do not

Disadvantage

- can be ***slow***

Examples

1. LR, SVM, nearest neighbors, least squares, …
2. holdout error, cross validation, bootstrap, …
3. Forward selection
   - start with no features
   - try adding each one, one at a time
   - pick the best, and then repeat

   Backward elimination
   - start with all features
   - try removing each one, one at a time
   - remove the worst, and then repeat

   Many, many others (see "greedy algorithms for sparse recovery" for hundreds of examples)

Embedded methods

Embedded methods ***jointly*** perform feature selection and model fitting instead of dividing these into two separate processes

The idea is to simultaneously learn a classifier or regression function that does well on the training data while only using a small number of features

Prime examples:

- LASSO
- Any other learning algorithm that uses ℓ_1-norm regularization

Feature extraction

In general, there may not be a small subset of features that works well

Examples

- speech
- images
- almost any sampled signal

How can we design a good mapping $\mathbf{x} \rightarrow \boldsymbol{\theta}$ that minimizes the loss of information using only the data we are given?

We will approach this from an unsupervised perspective

Principal component analysis (PCA)

- Unsupervised
- Linear
- Loss criteria: Sum of squared errors

The idea behind PCA is to find an approximation

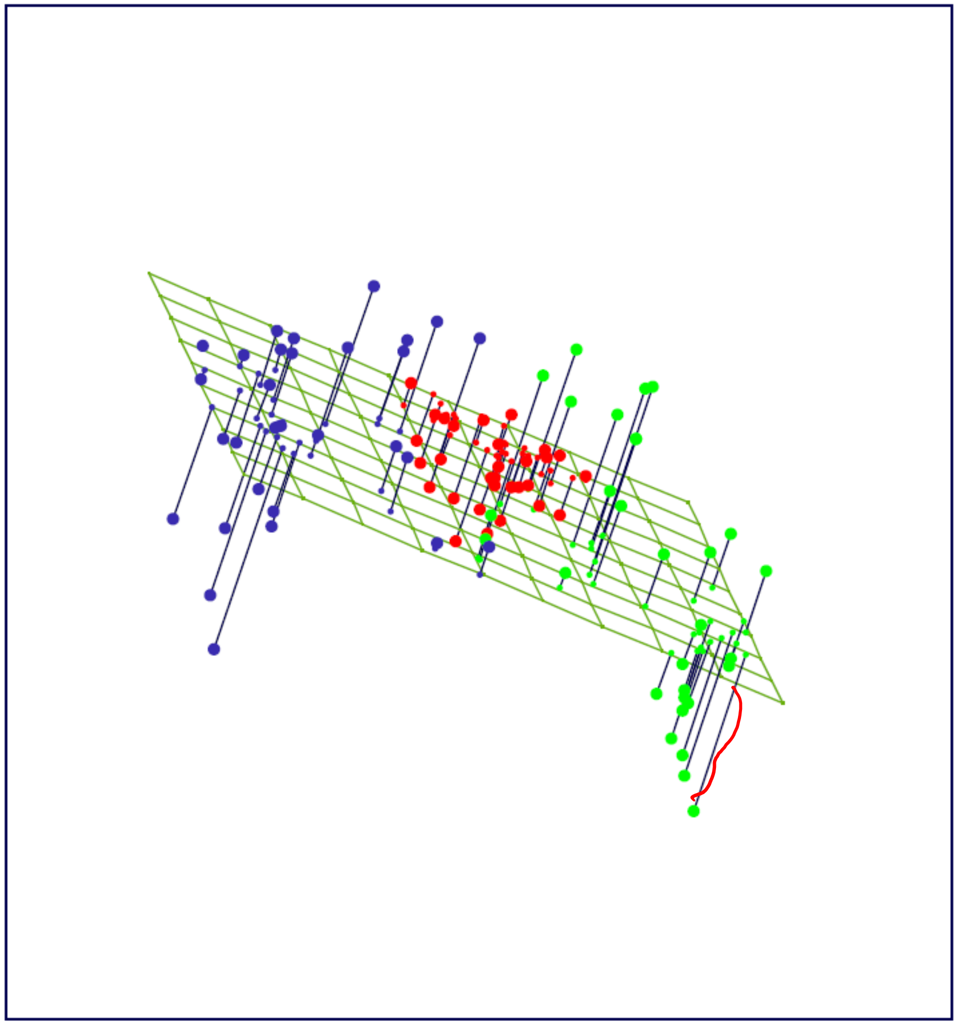
$$\mathbf{x}_i \approx \boldsymbol{\mu} + \mathbf{A}\boldsymbol{\theta}_i$$

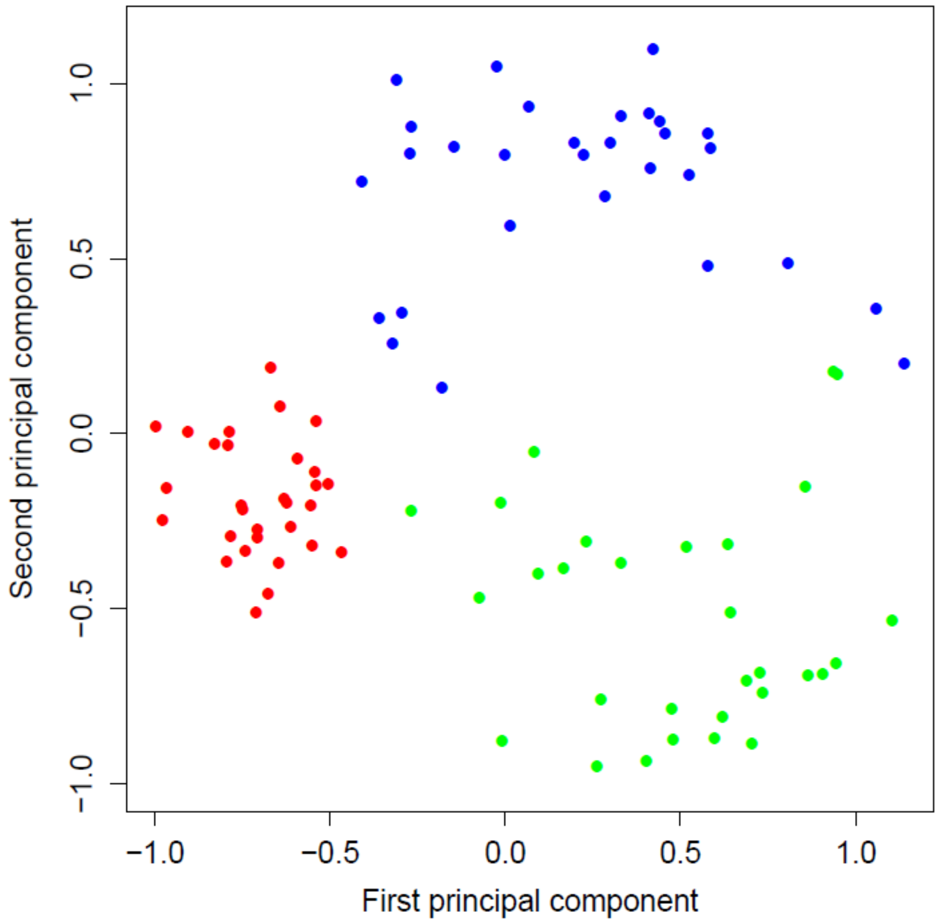
where

- $\boldsymbol{\mu} \in \mathbb{R}^d$
- $\mathbf{A} \in \mathbb{R}^{d \times k}$ with orthonormal columns
- $\boldsymbol{\theta}_i \in \mathbb{R}^k$

Example

From Chapter 14 of Hastie, Tibshirani, and Friedman





Derivation of PCA

Mathematically, we can define $\boldsymbol{\mu}, \mathbf{A}$ and $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n$ as the solution to

$$\min_{\boldsymbol{\mu}, \mathbf{A}, \{\boldsymbol{\theta}_i\}} \quad \sum_{i=1}^{n} \|\mathbf{x}_i - \boldsymbol{\mu} - \mathbf{A}\boldsymbol{\theta}_i\|_2^2$$

The hard part of this problem is finding $\mathbf{A}$

Given $\mathbf{A}$, it is relatively easy to show that

$$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$$

$$\boldsymbol{\theta}_i = \mathbf{A}^T(\mathbf{x}_i - \boldsymbol{\mu})$$

Determining $\boldsymbol{\theta}_i$

Suppose $\boldsymbol{\mu}, \mathbf{A}$ are fixed. We wish to minimize

$$\sum_{i=1}^{n} \|\mathbf{x}_i - \boldsymbol{\mu} - \mathbf{A}\boldsymbol{\theta}_i\|_2^2$$

Claim: We must have

$$\begin{aligned}\boldsymbol{\theta}_i &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T(\mathbf{x}_i - \boldsymbol{\mu}) \\ &= \mathbf{A}^T(\mathbf{x}_i - \boldsymbol{\mu})\end{aligned}$$

Why?

Determining $\boldsymbol{\theta}_i$ is just standard least-squares regression

Determining $\boldsymbol{\mu}$

Setting $\boldsymbol{\theta}_i = \mathbf{A}^T(\mathbf{x}_i - \boldsymbol{\mu})$ and still supposing $\mathbf{A}$ is fixed, our problem reduces to minimizing

$$\begin{aligned}
&\sum_{i=1}^{n} \|\mathbf{x}_i - \boldsymbol{\mu} - \mathbf{A}\mathbf{A}^T(\mathbf{x}_i - \boldsymbol{\mu})\|_2^2 \\
&\qquad = \sum_{i=1}^{n} \|(\mathbf{I} - \mathbf{A}\mathbf{A}^T)(\mathbf{x}_i - \boldsymbol{\mu})\|_2^2 \\
&\qquad = \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})^T \underbrace{(\mathbf{I} - \mathbf{A}\mathbf{A}^T)^T(\mathbf{I} - \mathbf{A}\mathbf{A}^T)}_{\mathbf{B}}(\mathbf{x}_i - \boldsymbol{\mu})
\end{aligned}$$

Determining $\boldsymbol{\mu}$

Taking the gradient with respect to $\boldsymbol{\mu}$ and setting this equal to zero, we obtain

$$-2\sum_{i=1}^{n}\mathrm{B}(\mathbf{x}_i - \boldsymbol{\mu}) = 0$$

$$\rightarrow \quad -2\mathrm{B}\left(\sum_{i=1}^{n}\mathbf{x}_i - n\boldsymbol{\mu}\right) = 0$$

The choice of $\boldsymbol{\mu}$ is not unique, but the easy (and standard) way to ensure this equality holds is to set

$$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$$

Determining $\mathbf{A}$

It remains to minimize

$$\sum_{i=1}^{n} \|\mathbf{x}_i - \boldsymbol{\mu} - \mathbf{A}\mathbf{A}^T(\mathbf{x}_i - \boldsymbol{\mu})\|_2^2$$

with respect to $\mathbf{A}$

For convenience, we will assume that $\boldsymbol{\mu} = \mathbf{0}$, since otherwise we could just substitute $\widetilde{\mathbf{x}}_i = \mathbf{x}_i - \boldsymbol{\mu}$

In this case the problem reduces to minimizing

$$\sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{A}\mathbf{A}^T\mathbf{x}_i\|_2^2$$

Determining $\mathbf{A}$

Expanding this out, we obtain

$$\begin{aligned}\sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{A}\mathbf{A}^T\mathbf{x}_i\|_2^2 &= \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{A}\mathbf{A}^T\mathbf{x}_i)^T(\mathbf{x}_i - \mathbf{A}\mathbf{A}^T\mathbf{x}_i) \\ &= \sum_{i=1}^{n} \mathbf{x}_i^T\mathbf{x}_i - 2\mathbf{x}_i^T\mathbf{A}\mathbf{A}^T\mathbf{x}_i + \mathbf{x}_i^T\mathbf{A}\underbrace{\mathbf{A}^T\mathbf{A}}_{\mathbf{A}^T\mathbf{A} = \mathbf{I}}\mathbf{A}^T\mathbf{x}_i \\ &= \sum_{i=1}^{n} \mathbf{x}_i^T\mathbf{x}_i - \mathbf{x}_i^T\mathbf{A}\mathbf{A}^T\mathbf{x}_i\end{aligned}$$

Thus, we can instead focus on maximizing

$$\sum_{i=1}^{n} \mathbf{x}_i^T\mathbf{A}\mathbf{A}^T\mathbf{x}_i$$

Determining $\mathbf{A}$

Note that for any vector $\mathbf{v}$, we have $\|\mathbf{v}\|_2^2 = \text{trace}(\mathbf{v}\mathbf{v}^T)$

Thus, we can write

$$\begin{aligned}\sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{A}\mathbf{A}^T \mathbf{x}_i &= \sum_{i=1}^{n} \|\mathbf{A}^T \mathbf{x}_i\|_2^2 \\ &= \sum_{i=1}^{n} \text{trace}(\mathbf{A}^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{A}) \\ &= \text{trace}\left(\mathbf{A}^T (\textstyle\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T) \mathbf{A}\right) \\ &= \text{trace}\left(\mathbf{A}^T \mathbf{S} \mathbf{A}\right)\end{aligned}$$

$\mathbf{S} = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T$ is a scaled version of the empirical covariance matrix, sometimes called the ***scatter*** matrix

Determining $\mathbf{A}$

The problem of determining $\mathbf{A}$ reduces to the optimization problem

$$\max_{\mathbf{A}} \ \text{trace}(\mathbf{A}^T\mathbf{S}\mathbf{A})$$
$$\text{s.t.} \ \mathbf{A}^T\mathbf{A} = \mathbf{I}$$

Analytically deriving the optimal $\mathbf{A}$ is not too hard, but is a bit more involved than you might initially expect (especially if you already know the answer)

We will provide justification for the solution for the $k = 1$ case – the general case is proven in the supplementary notes

One-dimensional example

Consider the optimization problem

$$\max_{\mathbf{a}} \ \mathbf{a}^T \mathbf{S} \mathbf{a}$$

$$\text{s.t. } \mathbf{a}^T \mathbf{a} = 1$$

Form the Lagrangian $\mathcal{L}(\mathbf{a}) = \mathbf{a}^T \mathbf{S} \mathbf{a} + \lambda(\mathbf{a}^T \mathbf{a} - 1)$

Take the gradient and set it equal to zero

$$\mathbf{S}\mathbf{a} + \lambda \mathbf{a} = 0$$

$\Rightarrow$ $\mathbf{a}$ must be an eigenvector of $\mathbf{S}$

Take $\mathbf{a}$ to be the eigenvector of $\mathbf{S}$ corresponding to the maximal eigenvalue

The general case

For general values of k, the solution is obtained by computing the eigendecomposition of $\mathbf{S}$:

$$\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$$

where $\mathbf{U}$ is an orthonormal matrix with columns $\mathbf{u}_1, \ldots, \mathbf{u}_d$ and

$$\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_d \end{bmatrix}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$

The general case

The optimal choice of $\mathbf{A}$ in this case is given by

$$\mathbf{A} = [\mathbf{u}_1, \ldots, \mathbf{u}_k]$$

i.e., take the top k eigenvectors of $\mathbf{S}$

Terminology

- principal component transform: $\mathbf{x} \to \boldsymbol{\theta} = \mathbf{A}^T(\mathbf{x} - \boldsymbol{\mu})$
- j^{th} principal component: $\theta(j) = \mathbf{u}_j^T(\mathbf{x} - \boldsymbol{\mu})$
- j^{th} principal eigenvector: $\mathbf{u}_j$